IMPROVED MICROSEISMIC SIGNAL DETECTION USING WEIGHTED PARTICLE MOTION LINEARITY MEASURES

A Thesis

by

ABDULLAH ABDULAZIZ A AHMED

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Richard Gibson
Committee Members,	Benchun Duan
	Eduardo Gildin
Head of Department,	Michael Pope

August 2018

Major Subject: Geophysics

ABSTRACT

Microseismic data provides important information about the subsurface during hydraulic fracturing jobs. However, microseismic signals are not as easy to identify as conventional seismic signals. I developed a new technique to detect microseismic signals by measuring displacement information along with the three-dimensional time coherence analysis of the 3D particles motion (linearity). It was tested on microseismic data from a horizontal well in the Marcellus Shale. The method multiplies the linearity calculation with a function of the amplitude of oscillation that is generated using the envelope. In addition, I developed techniques to detect signals from the new, weighted linearity (or the traditional, unweighted linearity) that help with the goal of more effective signal detection. The results from the new method show that it can detect 16 signals from two microseismic datasets, including barely noticeable (amplitude of 0.0003), weak (amplitude of 0.002), and strong (amplitude of 0.02) signals. These were compared with the results from the traditional, unweighted linearity calculation, where I detect only 9 signals and give results contaminated with noise. This indicates that there is a 40% improvement in signal detection using the new approach. Furthermore, the weighted linearity showed more detail in the signals and less noise compared to unweighted linearity results. With this new approach, I am able to detect signals that unweighted linearity cannot identify, while not compromising the quality of signals detectable by linearity.

DEDICATION

To my mother, father, and my lovely wife, for all their love and support.

ACKNOWLEDGMENTS

I would like to thank my committee chair Professor Richard Gibson for his guidance, support, encouragement, and making me a better scientist throughout the course of this research. I also like to acknowledge my committee members, Professors Benchun Duan and Eduardo Gildin for their feedback and support. I am also grateful for INTviewer and GeoTomo who provided the software.

Thanks also to my friends, colleagues, and the department faculty and staff for making my time at Texas A&M University a great experience. I also want to extend my gratitude to Saudi Aramco, which sponsored me as a M.S. student.

Finally, a special thanks has to go to my mother and father for their encouragement and to my wife for her patience and love.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a thesis committee consisting of Professor Richard Gibson [chair advisor] and Professor Benchun Duan of the Department of Geology & Geophysics and Professor Eduardo Gildin of the Department of Petroleum Engineering.

The data analyzed for Chapter 4 was provided by Halliburton.

All other work conducted for the thesis was completed by the student independently.

Funding Sources

Graduate study was supported by a sponsorship from Saudi Aramco.

NOMENCLATURE

3D	Three-dimensional
Cp	Global polarization
λ	eigenvalues
N	Total number of samples
V	Variance-Covariance matrix
P	eigenvectors of the matrix V
d	Displacement
ns	Number of Samples
H	Hilbert Transform
f	Frequency
m	Meters
ms	Millisecond

TABLE OF CONTENTS

Page

ABSTRACT ii

DEDICATION iii

ACKNOWLEDGMENTS iv

CONTRIBUTORS AND FUNDING SOURCES v

NOMENCLATURE vi

TABLE OF CONTENTS vii

LIST OF FIGURES ix

1. INTRODUCTION 1

2. DATA 5

3. METHOD 7

    3.1 Linearity Calculation 7
    3.2 Displacement Measurements 9
    3.3 Weighted Linearity Method 12
    3.4 Signal Detection Techniques 12

4. RESULTS 15

    4.1 Perforation Data 15
    4.2 Fracturing Data 16
        4.2.1 Window 1 (0 s - 0.5 s) 16
        4.2.2 Window 3 (2.4 s - 3 s) 17
    4.3 Cutoff Value Estimation 17

5. DISCUSSIONS 24

    5.1 Perforation Result 24
    5.2 Fracturing Results 25
        5.2.1 Window 1 (0 s - 0.5 s) 25
        5.2.2 Window 3 (2.4 s - 3 s) 26
    5.3 General Discussion 27

6. CONCLUSION 30

REFERENCES 31

APPENDIX A. COMPARISON BETWEEN THE FOUR APPROACHES OF DISPLACEMENT MEASUREMENTS 34

APPENDIX B. HILBERT TRANSFORM IN FREQUENCY DOMAIN 37

APPENDIX C. RESULTS AND DISCUSSIONS 39

C.1 Perforation Data 39
C.2 Fracturing Data 39
C.2.1 Window 1 (0 s - 0.5 s) 39
C.2.2 Window 2 (1.4 s - 2 s) 40
C.2.3 Window 3 (2.4 s - 3 s) 43
C.2.4 Window 4 (3 s - 3.5 s) 44
C.2.5 Window 5 (3.5 s - 5 s) 47

APPENDIX D. WEIGHTED AND UNWEIGHTED LINEARITY CODE 51

APPENDIX E. SIGNAL DETECT CODE 59

LIST OF FIGURES

FIGURE Page

1.1 Schematic showing the effect of the simultaneous arrival of two signals with different linear polarization directions, resulting in an observed signal with circular polarization. 3

1.2 Left panel: shows the vertical component of data from a perforation event in stage 2. The data was filtered using a bandpass filter with frequencies of 5 Hz, 10 Hz, 250 Hz, and 300 Hz. The twelve traces represent the number of geophones in the monitoring well. Right panel: shows the result of linearity calculation from the unfiltered data, with the red box indicating the arrival times of the S wave. This shows that the S wave arrives within the red box in the linearity are not clear, while they are in the filtered data (left panel). 4

2.1 Left panel: a map view of wells 1H, 2H, and the monitoring well (1V) at the Marcellus Shale formation. The different colors represent the fracturing stages, and stage 1 in well 1H is represented by the red line (top right). Right panel: Lateral side view of the wells, and the green squares inside well 1V represent the 12 Geophones. The Figure courtesy of Halliburton [1] 6

3.1 Left panel: shows the vertical component of data from a perforation event in stage 2. The data was filtered using a bandpass filter with frequencies of 5 Hz, 10 Hz, 250 Hz, and 300 Hz. Middle panel: shows the result of linearity calculation using the filtered data (left panel). The plot has global scaling between 0 and 1. Right panel: shows the result of linearity calculation using the filtered data (left panel). the plot has global scaling between 0 and 1. In addition, the black shaded peaks represent linearity values over 0.75 9

3.2 Panel 1: shows the vertical component of data from a fracturing event in stage 1. The data was filtered using a bandpass filter with frequencies of 5 Hz, 10 Hz, 250 Hz, and 300 Hz. Panel 2: shows the new modified linearity result, and the plot has global scaling between 0 and 0.3. Panel 3: shows the ratios of the average new weighted linearity (panel 2) between two moving windows with size of 7ms. The plot has global scaling between 0 and 4. Panel 4: shows the result of the result of the second technique of signal detection, which applies a cutoff value of 2.2, and the plot has global scaling between 0 and 1. Panel 5: shows the curve that represent the stack of panel 4. Panel 6: shows the end result of the third technique of signal detection after filtering out noise away from the signals using the stack curve in panel 5. The plot has global scaling between 0 and 1. 14

4.1 Left panels: shows the vertical component of filtered data from a perforation event in stage 2. A bandpass filter with filter parameters of 10 Hz, 20 Hz, 250 Hz, and 350 Hz, was applied. Figure C.1 in appendix C shows the three components of this event. Middle panels: a) shows the result from the unweighted linearity calculation. The plot has global scaling between 0 and 1. The black shaded peaks have linearity values above 0.5. b) shows the result from the new weighted linearity calculation. The plot has global scaling between 0 and 0.3. Right panels: show the result of detecting signal from the middle panels, where the peaks (signals) have a value of 1 and the rest are zeros. The plot has global scaling between 0 and 1, and the black shades represent values above 0.5. 18

4.2 Left panel: shows the result of calculating the ratios of the average unweighted linearity between two moving windows with sizes of 7 ms. Right panel: shows the result of calculating the ratios of the average weighted linearity between two moving windows with sizes of 7 ms. The plots have global scaling between 0 and 4. The black shades represent peaks with ratios above 2.5. 19

4.3 Shows time windows used in further analysis. Data is a fracturing file from stage 1. The data was filtered using a bandpass filter with frequencies of 5 Hz, 10 Hz, 250 Hz, and 300 Hz. The red waveform represents the vertical component (Z), the blue waveform represents the X component, and the green waveform represents the Y component. 19

4.4 Left panels: shows the vertical component of unfiltered data from window 1 Figure 4.3. Figure C.2 in appendix C shows the three components of this event. Middle panels: a) shows the result of the unweighted linearity calculation, and the plot has global scaling between 0 and 1. b) shows the result of the weighted linearity calculation. The plot has global scaling between 0 and 0.3. Right panels: show the result of detecting signal from the middle panels, where the peaks (signals) have a value of 1 and the rest are zeros. The plot has global scaling between 0 and 1. The black shades represent peak with values higher than 0.5. 20

4.5 Left panel: shows the result of calculating the ratios of the average unweighted linearity in window 1 between two moving windows with sizes of 7 ms. Right panel: shows the result of calculating the ratios of the average weighted linearity in window 1 between two moving windows with sizes of 7 ms. The plots have global scaling between 0 and 4. The black shades represent peaks with ratios above 2.5. ... 21

4.6 Left panels: shows the vertical component of unfiltered data from window 3 Figure 4.3. Figure C.7 in appendix C shows the three components of this event. Middle panels: a) shows the result of the unweighted linearity calculation, and the plot has global scaling between 0 and 1. b) shows the result of the weighted linearity calculation. The plot has global scaling between 0 and 0.3. Right panels: show the results of detecting signal from the middle panels, where the peaks (signals) have a value of 1 and the rest are zeros. The plots have global scaling between 0 and 1. The black shades represent peak with values higher than 0.5. 22

4.7 Left panel: shows the result of calculating the ratios of the average unweighted linearity in window 3 between two moving windows with sizes of 7 ms. Right panel: shows the result of calculating the ratios of the average weighted linearity in window 3 between two moving windows with sizes of 7 ms. The plots have global scaling between 0 and 4. The black shades represent peaks with ratios above 2.5. ... 23

4.8 Shows the relationship between the cutoff value and the average ratios of weighted linearity. The blue dots represent the data from each of the 6 data files. 23

5.1 Show the signal detection result from the new weighted linearity method of a) the perforation example (right panel, Figure 4.1(b)), b) window 1 (right panel, Figure 4.4(b)), and c) window 3 (right panel, Figure 4.6(b)). The blue, red, green and magenta curves represent the P, P coda, S, and S coda waves, respectively. 28

5.2 Shows a fracturing file from early stage 1. The red waveform represents the vertical component (Z), the blue waveform represents the X component, and the green waveform represents the Y component. 29

5.3 Left panel: shows the vertical component of the unfiltered data in Figure 5.2. Middle panel: shows the result of the weighted linearity method. The plot has global scaling between 0 and 0.3. Right panel: shows the result of detecting signal from the new, weighted linearity approach (middle panel). The plot has global scaling between 0 and 1. 29

A.1 Panel 1: shows the vertical component of data from a perforation event in stage 2. The data was filtered using a bandpass filter with frequencies of 0 Hz, 10 Hz, 250 Hz, and 300 Hz. Panel 2: shows the resulted envelope from panel (1). Panel 3: shows the unweighted linearity calculation from panel (1). The plot has global scaling between 0 and 1. Panels 4: displays the result of the first approach, which multiplies Panel (3) by equation (3.8). Panels 5: displays the result of the second approach, which multiplies Panel (3) by equation (3.9). Panels 6: displays the result of the third approach, which multiplies Panel (3) by equation (3.10). Panels 7: displays the result of the fourth approach, which multiplies Panel (3) by equation (3.11). The plots (4 through 7) have global scaling between 0 and 0.4. 35

A.2 Panel 1: shows the vertical component of data from a fracturing event in stage 1. The data was filtered using a bandpass filter with frequencies of 0 Hz, 10 Hz, 250 Hz, and 300 Hz. Panel 2: shows the resulted envelope from panel (1). Panel 3: shows the unweighted linearity calculation from panel (1). The plot has global scaling between 0 and 1. Panels 4, and 5: shows the result from multiplying panel (3) with the third approach and fourth approach, respectively. The plots have global scaling between 0 and 0.4. 36

B.1 Left panel: Shows a schematic example of a $cosine$ function in the frequency domain that has a range of (0,N). Right panel: shows an equivalent display of the function in the left panel 38

C.1 shows a perforation file from stage 2. The event was filter by a bandpass filter with frequencies of 5 Hz, 10 Hz, 250 Hz, and 300 Hz. The red waveform represents the vertical component (Z), the blue waveform represents the X component, and the green waveform represents the Y component. 39

C.2 shows a zoomed plot of Figure 4.3 at window 1. The different waveformsâĂŹ color represents the different component of a geophone, where red, blue and green are Z, X, and Y, respectively. 40

C.3 Shows a zoomed plot of Figure 4.3 at window 2. The different waveforms' color represents the different component of a geophone, where red, blue and green are Z, X, and Y, respectively. 40

C.4 Left panels: show the vertical component of unfiltered data from window 2 Figure 4.3. Middle panels: a) shows the result of the unweighted linearity calculation, and the plot has global scaling between 0 and 1. b) shows the result of the weighted linearity calculation. The plot has global scaling between 0 and 0.3. Right panels: show the result of detecting signal from the middle panels, where the peaks (signals) have a value of 1 and the rest are zeros. The plots have global scaling between 0 and 1. The black shades represent peak with values higher than 0.5. 41

C.5 Left panel: shows the result of calculating the ratios of the average unweighted linearity in window 2 between two moving windows with sizes of 7 ms. Right panel: shows the result of calculating the ratios of the average weighted linearity in window 2 between two moving windows with sizes of 7 ms. The plots have global scaling between 0 and 4. The black shades represent peaks with ratios above 2.5..... 42

C.6 Shows the signal detection result from the new weighted linearity method (middle panel, Figure C.4(b)). The blue, green, and magenta curves represent the P, S, and S coda waves, respectively. 43

C.7 shows a zoomed plot of Figure 4.3 at window 3. The different waveforms' color represents the different component of a geophone, where red, blue and green are Z, X, and Y, respectively. 44

C.8 Shows a zoomed plot of Figure 4.3 at window 4. The different waveforms' color represents the different component of a geophone, where red, blue and green are Z, X, and Y, respectively. 44

C.9 Left panels: show the vertical component of unfiltered data from window 4 Figure 4.3. Middle panels: a) shows the result of the unweighted linearity calculation, and the plot has global scaling between 0 and 1. b) shows the result of the weighted linearity calculation. The plot has global scaling between 0 and 0.3. Right panels: show the result of detecting signal from the middle panels, where the peaks (signals) have a value of 1 and the rest are zeros. The plots have global scaling between 0 and 1. The black shades represent peak with values higher than 0.5. 45

C.10 Left panel: shows the result of calculating the ratios of the average unweighted linearity in window 4 between two moving windows with sizes of 7 ms. Right panel: shows the result of calculating the ratios of the average weighted linearity in window 4 between two moving windows with sizes of 7 ms. The plots have global scaling between 0 and 4. The black shades represent peaks with ratios above 2.5..... 46

C.11 Shows the signal detection result from the new weighted linearity method (middle panel, Figure C.9(b)). The blue, green, and magenta curves represent the P, S, and S coda waves, respectively. 47

C.12 Shows a zoomed plot of Figure 4.3 at window 5. The different waveforms' color represents the different component of a geophone, where red, blue and green are Z, X, and Y, respectively. 48

C.13 Left panels: show the vertical component of unfiltered data from window 5 Figure 4.3. Middle panels: a) shows the result of the unweighted linearity calculation, and the plot has global scaling between 0 and 1. b) shows the result of the weighted linearity calculation. The plot has global scaling between 0 and 0.3. Right panels: show the result of detecting signal from the middle panels, where the peaks (signals) have a value of 1 and the rest are zeros. The plots have global scaling between 0 and 1. The black shades represent peak with values higher than 0.5. 49

C.14 Left panel: shows the result of calculating the ratios of the average unweighted linearity in window 5 between two moving windows with sizes of 7 ms. Right panel: shows the result of calculating the ratios of the average weighted linearity in window 5 between two moving windows with sizes of 7 ms. The plots have global scaling between 0 and 4. The black shades represent peaks with ratios above 2.5..... 50

1. INTRODUCTION

Microseismic data analysis is a topic of interest in many areas of geophysics. These seismic signals are induced by temporal changes in stress fields within the Earth's subsurface, such as those associated with earthquakes, fluid injection, and oil and gas production [2, 3]. The demand for microseismic monitoring has increased rapidly in recent years due to the increase in exploration for unconventional resources, high demand of hydrocarbons, and earthquake forecasting [4, 5]. Microseismic data is now recorded continuously, and this type of data provides a high chance to detect small events that cannot be seen by the conventional amplitude based triggered recording system [6]. Monitoring both small and large subsurface events are essential for understanding the reservoir system and stimulation process, evaluating the hydraulic fracturing injection program, focusing the drilling process, and increasing the production volume [7, 8].

Detecting signals, when incorporated with a velocity model, is important for event location (monitoring microseismic) and other applications. The main objective of signal detection is to be able to differentiate between signal arrivals and noise (e.g. multiples, ground roll, etc.). However, microseismic primary signals and reflections are not always easy to identify (e.g. S waves in perforation events, P waves in fracture events) due to low signal to noise ratio and the relatively small amplitudes compared to conventional seismic events [9, 10, 11]. This often leads to either missing whole events or mislocating them due to inaccurate time picks. For example, in this study it was seen that miss-picking an event by 1 milliseconds leads to a minimum of 5 meters mislocation, therefore, an accurate signal pick is essential to locate events.

There are multiple techniques used on continuously recorded microseismicity data to detect events [12], such as using displacement amplitudes, short-term-long-term average ratio (STA/LTA) [13], and trajectories of particle motions [14]. Specifically, application of linearity for signal detection was first introduced by Samson in 1977 in his signal detector design to analyze teleseismic waves in very efficient computational way in the frequency domain. Linearity has also been used to identify seismic reflections from deep geothermal reservoirs (e.g. [15, 16, 17]). However, these re-

flected signals were not clearly identifiable in the linearity results, so they performed a 3D inversion on these results to better detect deep subsurface structure of field from the linearity. Furthermore, linearity has recently been applied to microseismic data. For example, Mukuhira et al. (2017) used spectral matrix analysis of 3D particle motion in the time-frequency domain on microseismic. The linearity was evaluated in a frequency band with highest power to avoid noise. In addition, a statistical approach, which detect polarization variation between two moving windows, was used to identify P and S arrivals [6]

My research will be focusing on the linearity of particle motion, where coherent signals have linear particle motions and incoherent signals (noise) have circular motions [17, 14]. In addition, linearity increases quickly with the arrival of coherent waves on the three geophone components (e.g. signal) and decreases for incoherent noise [18].

There are many cases where linearity results show low values at signal arrival times where high values are expected. This occurs for multiple reasons. For example, different (linear) signals (e.g. P coda and S wave) could arrive at the exact same time but with a different polarization direction and phase, so they interfere with each other to produce a nonlinear behavior, as shown by the red circle in Figure 1.1. Different linear signals could also arrive at very close times leading to linearity results showing a broad window in time with high linearity values, which makes it difficult to differentiate the arrival signals in the linearity calculations that are clearly seen in the data, as shown by the red rectangle in Figure 1.2. I propose a new technique to resolve these issues by including displacement information with the linearity calculations, with the goal of more effective signal detection. An important challenge is that linearity is not directly related to the amplitude of a seismic signal. I will show that weighting the linearity result by an appropriately chosen measure of signal amplitude gives results that more effectively highlight individual arrivals, allowing for the development of a new method of signal detection using this modified linearity.

Below I first review the microseismic data examined in the paper. I then summarize the new event detection approach using weighted linearity. Finally, I apply the method to data and compare results to standard linearity to show the improvement in event detection derived in this study.

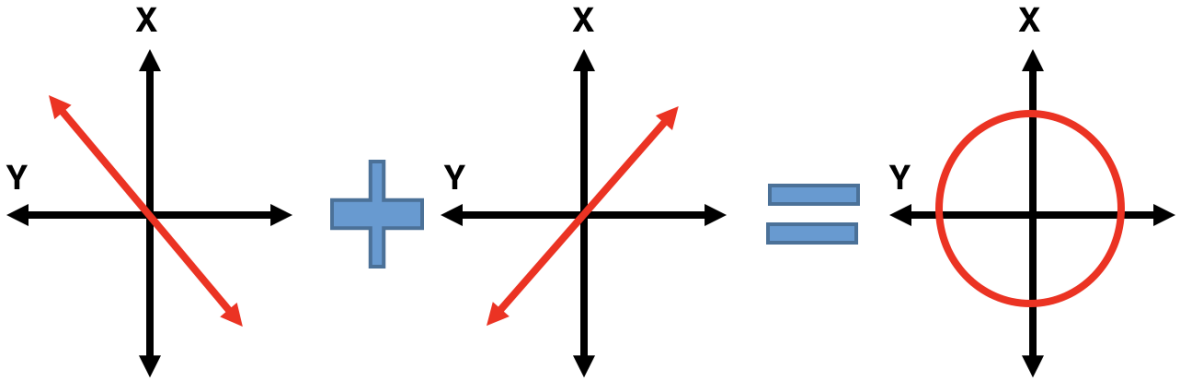


Figure 1.1: Schematic showing the effect of the simultaneous arrival of two signals with different linear polarization directions, resulting in an observed signal with circular polarization.

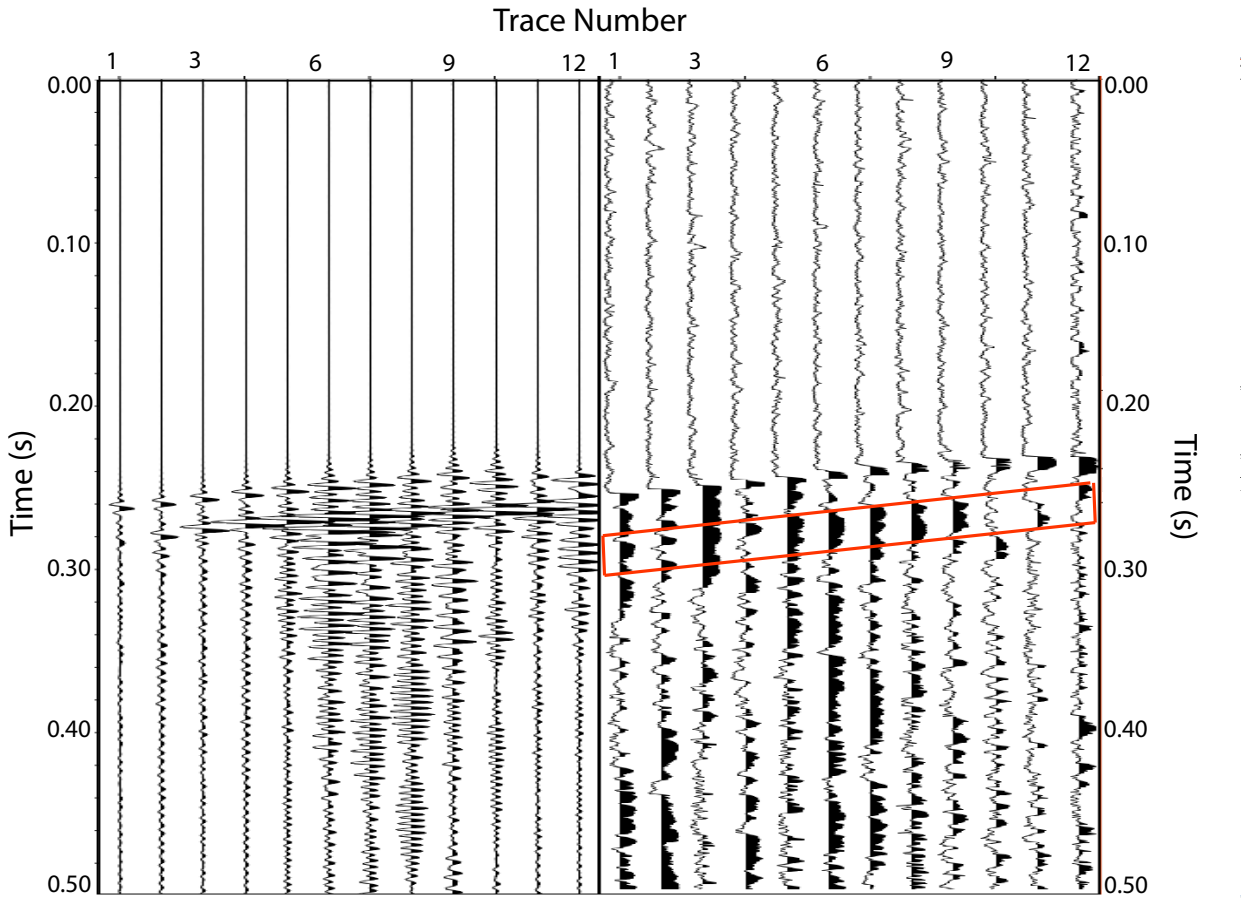


Figure 1.2: Left panel: shows the vertical component of data from a perforation event in stage 2. The data was filtered using a bandpass filter with frequencies of 5 Hz, 10 Hz, 250 Hz, and 300 Hz. The twelve traces represent the number of geophones in the monitoring well. Right panel: shows the result of linearity calculation from the unfiltered data, with the red box indicating the arrival times of the S wave. This shows that the S wave arrives within the red box in the linearity are not clear, while they are in the filtered data (left panel).

2. DATA

My research applies microcosmic data from two of Marcellus Shale horizontal wells. The two subject wells (Well 1H and Well 2H) are located in Tioga County, Pennsylvania [19]. Both wells were drilled horizontally approximately 61 m, and completed as cemented laterals in the objective Marcellus Shale formation (see figure 2.1). Figure 2.1 shows the seven stages of hydraulic fractures treatments (for gas production) using ”plug and perf” completion.

Well 1H was treated five weeks after well 2H, with different quantities of Slick-water, sand, and Jordan sand. There were five perforation clusters at each stage and each cluster has 0.3 m length [19]. The microseismic events were recorded at each stage (perforation and fracture events were recorded separately) by stationary 12 multi-component geophones that were located in the monitoring well (1V) shown in Figure 2.1. The monitoring well is a vertical well that is located in between the horizontal wells and very close to stage 2 in well 1H (see Figure 2.1). The lower 3 geophones are located in the Marcellus Shale formation and the rest are located in the above Skaneateles Shale member. In addition, the geophones vertical spacing is around 11 m, and the sampling rate is 0.25 ms. Halliburton provided velocity values with depths at the well (1V) that were used in their microseismic fracture mapping analysis [19]. I have incorporated these values as part of my locationing analysis. This project focuses on microseismic data from well 1H only.

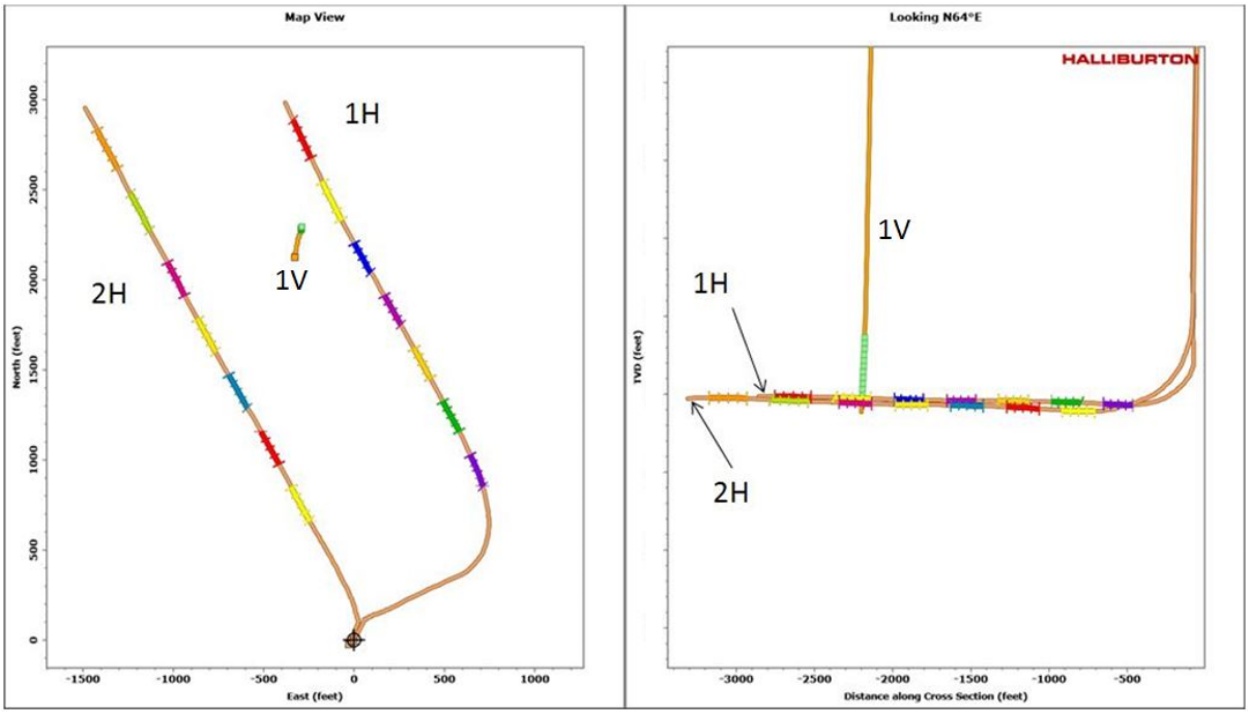


Figure 2.1: Left panel: a map view of wells 1H, 2H, and the monitoring well (1V) at the Marcellus Shale formation. The different colors represent the fracturing stages, and stage 1 in well 1H is represented by the red line (top right). Right panel: Lateral side view of the wells, and the green squares inside well 1V represent the 12 Geophones. The Figure courtesy of Halliburton [1]

3. METHOD

In this section I will show the steps for calculating the traditional linearity and the new weighted linearity approach introduced in this study. In addition, I describe the preferred technique to measure displacement information and will discuss the developed techniques for detecting signals.

3.1 Linearity Calculation

The linearity of the particle motion is represented by the global polarization coefficient (Cp) ranging from 0.00 and 1.00, where higher values (closer to 1) correspond to linear behavior and lower values represent non-linear behavior [17, 14]:

$$Cp = \frac{(\lambda 1 - \lambda 2)^2 + (\lambda 2 - \lambda 3)^2 + (\lambda 1 - \lambda 3)^2}{2 * (\lambda 1 + \lambda 2 + \lambda 3)^2}. \tag{3.1}$$

The values in the Cp equation are the eigenvalues of a variance-covariance matrix (V), which is obtained from discrete time series vectors (recorded signals on three components geophones (Sx, Sy, Sz)): [16]

$$V = \begin{pmatrix} Cxx & Cxy & Cxz \\ Cyx & Cyy & Cyz \\ Czx & Czy & Czz \end{pmatrix}, \tag{3.2}$$

where

$$Cij = \frac{\sum_{n=0}^{N} (Si(n) - \overline{Si}) * (Sj(n) - \overline{Sj})}{N}, \quad i, j = x, y, z, \tag{3.3}$$

$$u(t) = [Sx(t), Sy(t), Sz(t)], \tag{3.4}$$

$u(t)$ = recorded signal by a geophone and N = Total number of samples. The diagonal values of this matrix are the variance for each of the three directional components, and the off-diagonal values are the covariance between the directional components. The covariance matrix satisfies the

relationship in following equations: [16]

$$VP_i = \lambda_i P_i, \tag{3.5}$$

$$|V - \lambda_i I| = 0, \tag{3.6}$$

where p_i = eigenvectors of the matrix V, and λ_i = eigenvalues. The first eigenvalue represents the direction of polarization [17].

In this study, the elements of the variance-covariance matrix are only calculated in the time domain because all frequencies in the microseismic data (usually between 0 Hz and 1500 Hz) are needed in order to capture all present signals (weak, strong, and contaminated with noise), and the source-receiver distance is relatively small, so signals will be stronger and has lower noise compared to higher source-receiver distances. In contrast, most of the previous studies (e.g. [14, 20, 15]) used a spectral matrix (time and frequency domain, e.g. the first element Cxx (t,f)), where the shift of the calculation is in time and scale is in frequency. They used the spectral matrix due to source-receiver distance being large (e.g. reflection seismic from geothermal reservoirs). This allows them to choose certain frequencies (usually low frequencies, e.g. 0 Hz-20 Hz) that contain good signals and to eliminate the rest of the frequencies that contain noise.

I used a moving window method [21, 22, 23, 12] to calculate the global polarization coefficient as a function of time, which helps to detect different signals within a discrete time series vector. I choose the size of the moving window where the linearity is calculated as twice the period of a signal, and the time of the increment by which the processing window is moved is 1 sample.

I wrote a python code (Weighted and Unweighted linearity code), which can found in Appendix D, to calculate the linearity for the 12 three component (3C) Geophones used in this research.

After evaluating several linearity calculations, I found that it is optimal to calculate linearity from unfiltered data rather than filtered data because filtered data produces high linearity values at times where signals do not exist. For example, in Figure 3.1 the linearity values before the P arrivals are high in the middle panel (linearity from filtered data) since the frequencies below 10

Hz were filtered out, while the linearity values are close to zero at same times in the right panel (linearity from unfiltered data).

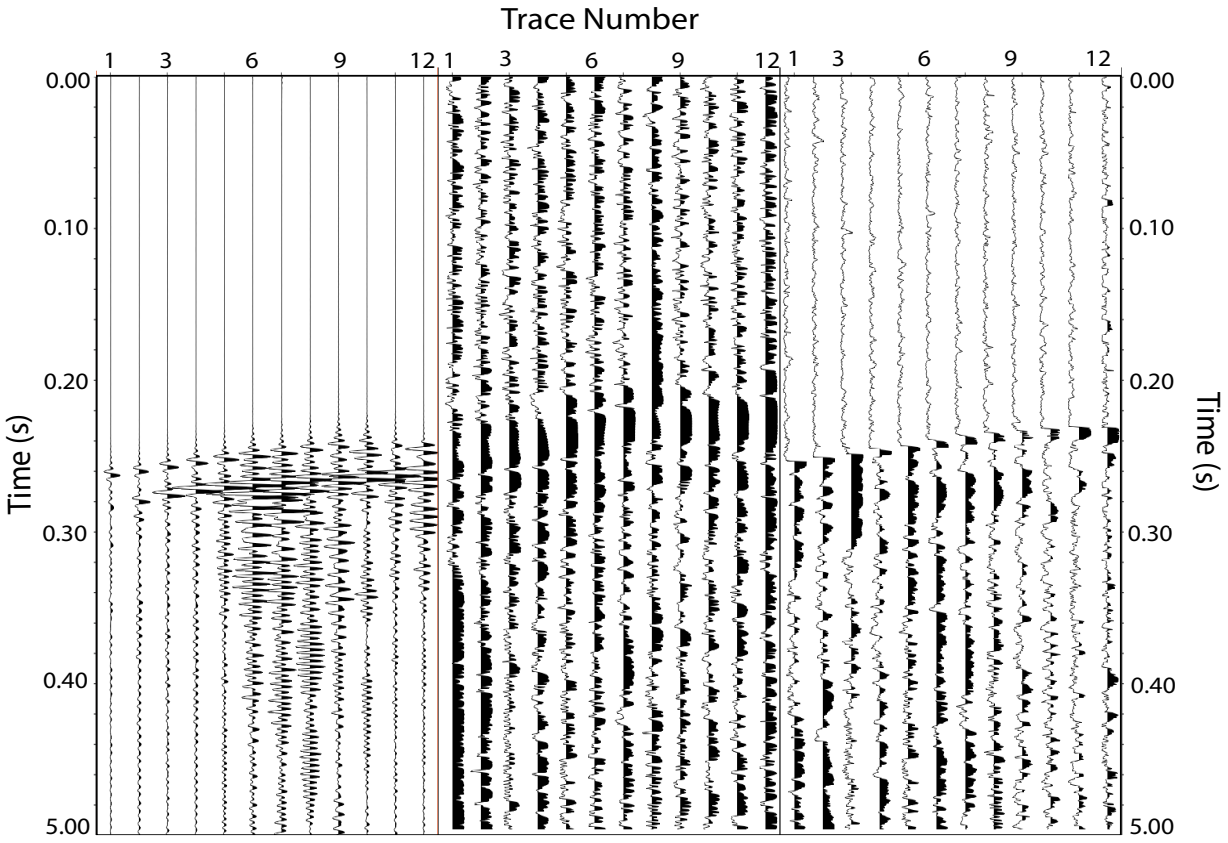


Figure 3.1: Left panel: shows the vertical component of data from a perforation event in stage 2. The data was filtered using a bandpass filter with frequencies of 5 Hz, 10 Hz, 250 Hz, and 300 Hz. Middle panel: shows the result of linearity calculation using the filtered data (left panel). The plot has global scaling between 0 and 1. Right panel: shows the result of linearity calculation using the filtered data (left panel). the plot has global scaling between 0 and 1. In addition, the black shaded peaks represent linearity values over 0.75

3.2 Displacement Measurements

The traditional, unweighted linearity can be improved by amplitude weighting to better identify signals in the cases where signals are arriving at the exact same time or very close times. In my displacement analysis, I seek to obtain the best possible estimate of the amplitude of oscillation

(d) at the time point at which the linearity is calculated. I have tested four different approaches of displacement measurements ($d1$ through $d4$) on perforation and fracturing data and compared them in the appendix A to determine the optimal method.

$$d = \sqrt{dx^2 + dy^2 + dz^2}, \tag{3.7}$$

$$d_1 = \sqrt{\frac{1}{ns}\sum_{i=1}^{i=ns}(d_i^2)}, \tag{3.8}$$

$$d_2 = \frac{1}{ns}\sum_{n=1}^{i=ns} d_i, \tag{3.9}$$

$$d_3 = \sqrt{\frac{1}{ns}\sum_{n=1}^{i=ns} d_i}, \tag{3.10}$$

$$d_4 = E(t) = \sqrt{u_i(t)^2 + H(u_i)(t)^2}, \tag{3.11}$$

where $E(t)$ = Envelope, ns = Number of samples, H = Hilbert Transform of the signal $u(t)$, and i = x,y,z.

I found that the amplitude of oscillation in a component is best estimated by using the envelope of a signal (equation 3.11). The envelope results (after being multiplied with linearity) show more sensitivity to low amplitudes and better signal representation compared with the other three displacement measurement methods (see appendix A). To calculate the envelope, a linear operator (Hilbert transform) is needed, which takes the time series function $u(t)$ with real variables (the recorded signal) and produces another function of a real variable $H(u)(t)$. It is defined by the following formula which is applied in the time domain [24]:

$$H(u)(t) = \frac{1}{\pi}\int_{-\infty}^{\infty}\frac{u(\tau)}{t-\tau}d\tau. \tag{3.12}$$

The Hilbert transform (equation 3.12) can also be defined as an operator that transforms cosine into sine, and sine into negative cosine. Using this information, I can express our signals as a

sum of sines and cosines, by using the Fourier transform and then apply the Hilbert transform to the trigonometric functions. The transformation of the trigonometric functions can be done in the frequency domain (after the Fourier transform is applied) by multiplying the frequencies ($Y(f)$) with imaginary numbers, as shown in following equation:

$$Y(f) * i \qquad for\ f \geq \frac{N}{2}, \tag{3.13}$$

$$Y(f) * -i \qquad for\ f \leq \frac{N}{2}. \tag{3.14}$$

A more detailed description of this process can be found in the appendix B. It is computationally efficient to perform the transformation in the frequency domain.

The python code (Weighted and Unweighted linearity code) (Appendix D) calculates the different amplitude measurements. I have used the moving window method as well for the first three approaches with size of a period of a signal. Furthermore, the code calculates the Hilbert transform in the frequency domain and uses the Fast Fourier Transform to decrease the computational time.

By measuring the envelope $E(t)$, I can get the amplitude of oscillation in each of the three components, however I am interested in the total amplitude of oscillation ($E_{Total}(t)$). To calculate this amplitude, I use:

$$E_{Total}(t) = \sqrt{E_X(t)^2 + E_Y(t)^2 + E_Z(t)^2}, \tag{3.15}$$

where $E_X(t)$ = total envelope at X direction, $E_Y(t)$ = total envelope at Y direction, and $E_Z(t)$ = total envelope at Z direction. In general, the envelope of the S wave is significantly larger than the envelope of the P wave. Therefore, I have taken the square root of the total envelope to reduce the measure for the high amplitudes of the S waves and to reduce the amplitude difference compared to P wave amplitudes. This helps in detecting the lower amplitudes of P waves. In addition, taking the square root increases the effect of the linearity in the multiplication process between the linearity and the envelope later on. This is because the amplitude values of the envelop are reduced by taking the square root. I refer this new envelope as the SQR envelope.

A peak of an envelope is often time at the middle of an arriving signal; however, the peak of

the linearity for the same signal will be at the beginning. This will have a distractive effect in the multiplication process later on. Therefore, a function proportional to the derivative of the envelope was added to the SQR envelope, because a sudden increase of envelope often indicates a wave arrival. Before being added to the SQR envelope, the function proportional to the derivative was scaled to match the average values of the SQR envelope. From now on, I refer to the sum of these two functions as the new modified envelope ($E_{Mod}(t)$), as presented in the following equation:

$$E_{Mod}(t) = \sqrt{E_{Total}(t)} + (\sqrt{E_{Total}(t)})' * \frac{average(|\sqrt{E_{Total}(t)}|)}{2 * average(|(\sqrt{E_{Total}(t)})'|)}, \tag{3.16}$$

where $\sqrt{E_{Total}(t)}'$ = the derivative of $\sqrt{E_{Total}(t)}$, $E_{Mod}(t)$ = the new modified envelope, and $E_{Total}(t)$ = total envelope.

3.3 Weighted Linearity Method

My new approach (weighted linearity) applies multiplication between the calculated linearity and the new modified envelope (equation 3.16).

3.4 Signal Detection Techniques

I have developed different methods to detect microseismic signals from the new, weighted linearity approach or the traditional, unweighted linearity approach. The first technique is using ratios of the average weighted linearity (or the unweighted linearity) between two moving windows (see panel 3, Figure 3.2). I have chosen the size of the windows as the period of a signal, and the time of the increment by which the processing windows are moved is 1 sample. The result is intended to show high ratios values at signal arrival, and low ratios everywhere else, as shown in panel 3 (Figure 3.2). The ratios will be calculated at the boundary time between the two moving windows; however, the signal will be at one of the windows. Therefore, a vertical time shift needs to be applied to the ratio results to match with the arrival signals. The amount of the time shift is determined by a cross correlation between the ratios result and the weighted linearity (or the unweighted linearity) result. This cross-correlation process is performed in the frequency domain

and the resulting 12 traces are stacked, with the maximum stack amplitude representing the time shift needed.

The second technique applies a cutoff value on the resulted ratio (first technique) to remove noise, and compares high peaks distribution in adjacent traces to remove isolated peaks that are produced by noise. In addition, points above the cutoff value are set to 1 and the rest are set to zero, as shown in panel 4 (Figure 3.2). This rescaling of amplitude values to be between 0 and 1 helps make a clear distinction between values above and below the cutoff value and serves my main goal, which is to detect signals accurately. The cutoff value can be detected automatically through a function that relates the average of the ratios and the cutoff values. In my data, I generated this function by measuring the optimum cutoff value for 6 data files and their average ratios, and fitted a straight line to the data.

The third method removes noise, which is left over from the second method, particularly the noise before and after signal arrival. It applies a stack on the result from the second technique (see panel 5, Figure 3.2), and then applies a filter that filters out any peak below a value that is five time smaller than the maximum peak at the stack curve in the result of the second technique. The end result of this method is shown in panel 6 (Figure 3.2).

The python code (Signal Detect) (Appendix E) applies all the three developed techniques to detect signals. For the second technique, the code was set to look for peaks in the ratios, and at each peak it will search for another peaks in the adjacent traces within a set window, which I set to be equal to twice of a period of the signal in my data.

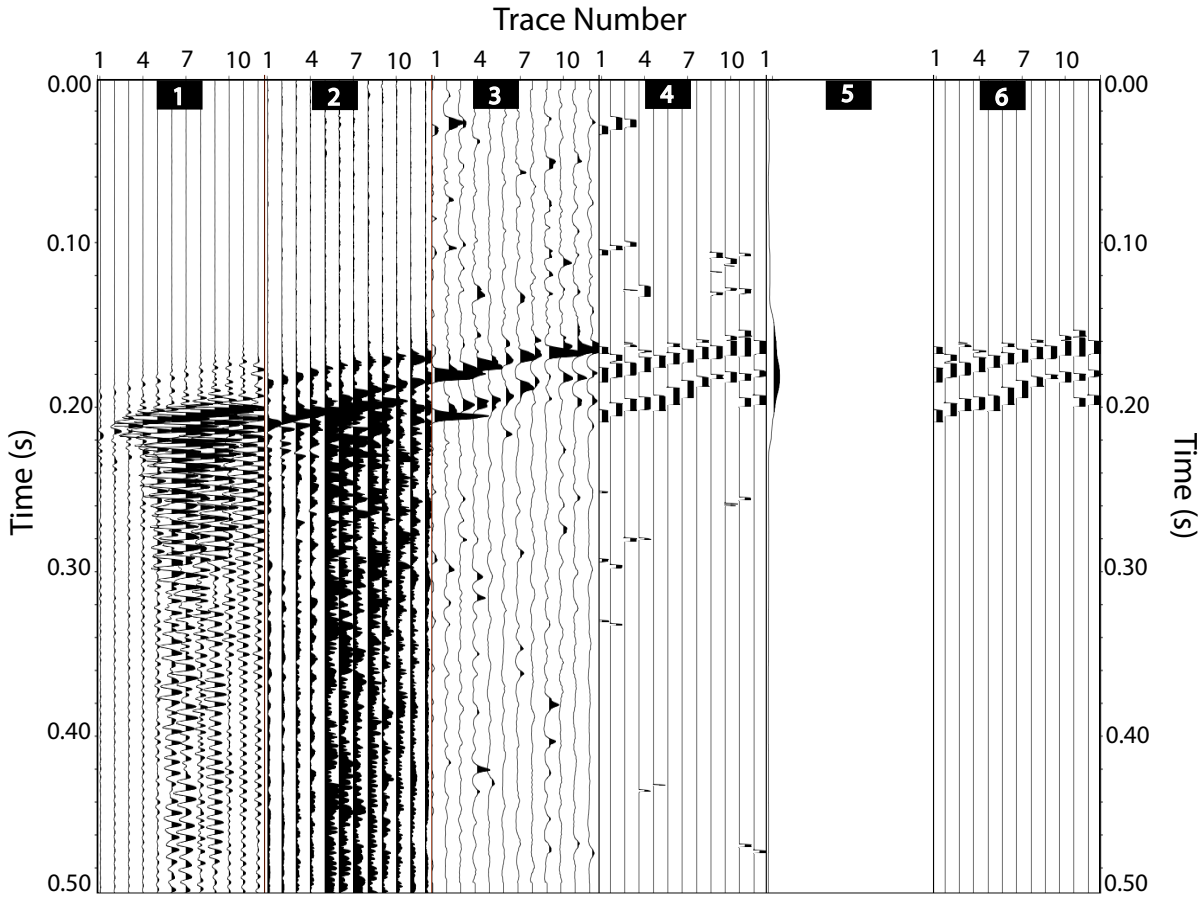


Figure 3.2: Panel 1: shows the vertical component of data from a fracturing event in stage 1. The data was filtered using a bandpass filter with frequencies of 5 Hz, 10 Hz, 250 Hz, and 300 Hz. Panel 2: shows the new modified linearity result, and the plot has global scaling between 0 and 0.3. Panel 3: shows the ratios of the average new weighted linearity (panel 2) between two moving windows with size of 7ms. The plot has global scaling between 0 and 4. Panel 4: shows the result of the result of the second technique of signal detection, which applies a cutoff value of 2.2, and the plot has global scaling between 0 and 1. Panel 5: shows the curve that represent the stack of panel 4. Panel 6: shows the end result of the third technique of signal detection after filtering out noise away from the signals using the stack curve in panel 5. The plot has global scaling between 0 and 1.

4. RESULTS

In this section, I apply the weighted linearity scheme developed in this study to the field data and compare it to the conventional linearity with no weighting. I chose two files from the field data to illustrate my new method. The first one is a perforation event, which has strong P and S arrivals. The second file is from the fracturing data, which includes signals from strong events, weak events, and barely noticeable events, allowing comparison of the linearity measures for different signal strengths.

4.1 Perforation Data

The 0.5 s data file in Figure 4.1(a) is a stage 2 perforation event, which has strong clear arrivals. A bandpass filter (frequencies: 5 Hz, 10 Hz, 250 Hz, and 300 Hz) was applied to better identify the P and S arrivals.

I first calculated the unweighted linearity with window size of 4 ms from the unfiltered data, with the result shown in middle panel of Figure 4.1(a). Then the new, weighted linearity was applied by multiplying the unweighted linearity (middle panel, Figure 4.1(a)) with the envelope calculated from the filtered data (left panel, Figure 4.1(b)), and the result is shown in the middle panel of Figure 4.1(b). The modified signal detection techniques were applied to the unweighted linearity and weighted linearity calculations. The ratios of the two moving windows with size of 7ms are shown in Figure 4.2 for both the weighted and unweighted linearity methods. A cutoff value of 2 was applied to the ratios of weighted linearity (right panel, Figure 4.2), and points above the cutoff value were set to 1 and the rest were set to zero. Furthermore, a lower cutoff value of 1.6 was applied to the ratios of the unweighted linearity (left panel, Figure 4.2) due to the ratios being 60% lower in the unweighted linearity compared to the weighted linearity ratios (see Figure 4.2). The final results for the signal detect techniques are shown in the right panels in Figures 4.1(a) and 4.1(b).

4.2 Fracturing Data

The 5 s file in Figure 4.3 includes events from stage 1 fracturing data. The different events present in the data have a range of different strengths (different amplitude values). For example, the strong event at 0.35 s has an average S wave amplitude of (0.02), the medium event at 0.15 s has an average S wave amplitude of 0.01, and the weak event at 1.6 s has an average S amplitude of 0.002. Furthermore, there are many barely noticeable events (e.g. at 1.45 s and 2.7 s) with an average S amplitude of 0.0003. The P waves are difficult to detect in the data for the medium, weak and barely noticeable events. The variety in event strength will help in the comparison process between the unweighted and weighted linearity approaches.

The unweighted linearity and weighted linearity schemes were applied to the unfiltered data with window size of 4 ms for the whole file. Furthermore, the signal detection code was applied to the results of the two methods. The size of the two windows, where ratios are calculated, is 7 ms. A cutoff value of 2.2 was applied to the resulting ratios of the weighted linearity, and a value of 1.7 was applied to the resulting ratios of the unweighted linearity method. Points above the cutoff values were set to 1 and the rest were set to zero.

Due to the length of the 5 s file, the result of the previous steps are displayed in separate figures, that have different time windows. Figure 4.3 shows the fracturing file with five different time windows, which include different events that will be further examined. In this paper, I chose windows 1 and 3 to illustrate my new, weighted linearity method. The rest of the windows are examined and discussed in appendix C.

4.2.1 Window 1 (0 s - 0.5 s)

The left panel in Figure 4.4(a) shows the first 0.5 s (window 1), which includes one of the medium events and the strongest event. The unweighted and weighted linearity result are shown in Figures 4.4(a) and 4.4(b) along with the final result of signal detection and the ratios plots are shown in Figure 4.5.

4.2.2 Window 3 (2.4 s - 3 s)

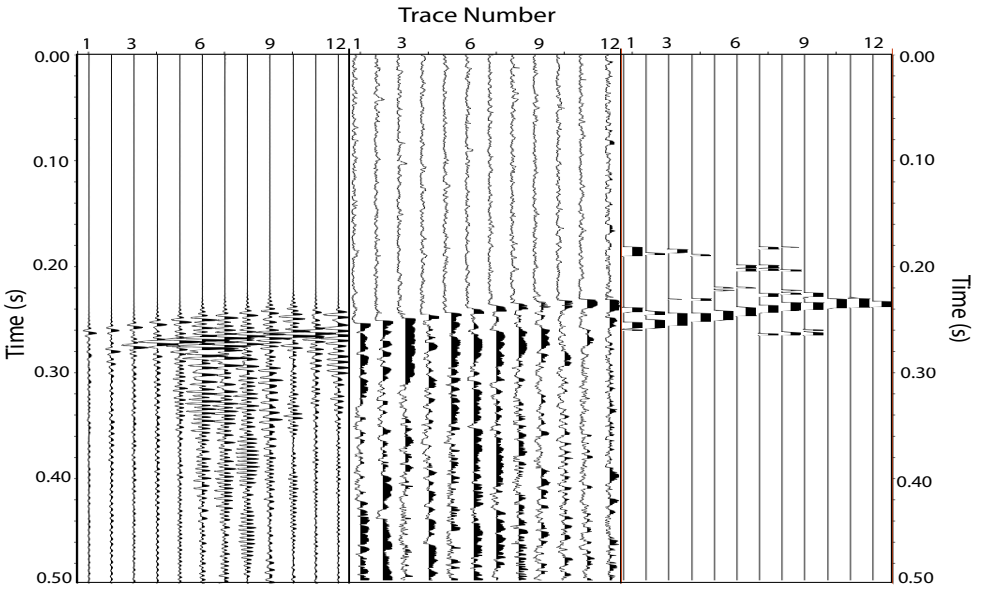
The left panel in Figure 4.6(a) shows a zoomed image of window 3, which includes a weak and barely noticeable events. The unweighted and weighted linearity result are shown in Figures 4.6(a) and 4.6(b) along with the final result of signal detection and the ratios plots are shown in Figure 4.7.

4.3 Cutoff Value Estimation

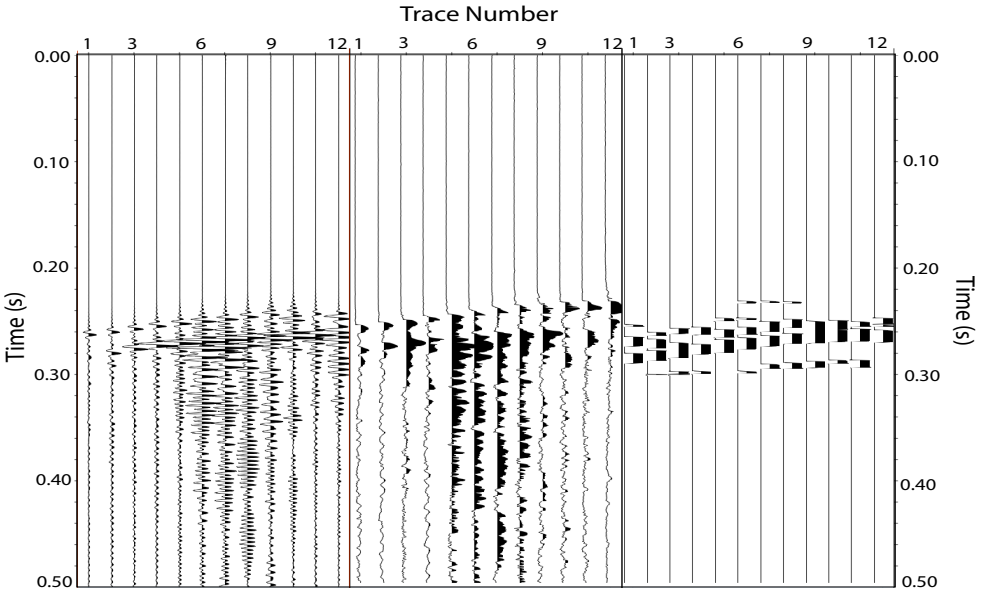
In the previous two examples, the cutoff values were estimated from the results of multiple tests. However, the project used in my research contains thousands of files and it is impossible to test the optimum value for each event; therefore, an automatic estimation of cutoff value is needed. A function, which was generated by fitting a straight line through 6 data points as shown in Figure 4.8, is used to estimate cutoff values (A_{cut}) from the calculated average ratios ($ratios_{ave}$) of linearity:

$$A_{cut} = 1.2894 * ratios_{ave} + 0.3875. \tag{4.1}$$

The data points were taken from 6 different data files (2 from perforation files and 4 from fracturing files).



(a)



(b)

Figure 4.1: Left panels: shows the vertical component of filtered data from a perforation event in stage 2. A bandpass filter with filter parameters of 10 Hz, 20 Hz, 250 Hz, and 350 Hz, was applied. Figure C.1 in appendix C shows the three components of this event. Middle panels: a) shows the result from the unweighted linearity calculation. The plot has global scaling between 0 and 1. The black shaded peaks have linearity values above 0.5. b) shows the result from the new weighted linearity calculation. The plot has global scaling between 0 and 0.3. Right panels: show the result of detecting signal from the middle panels, where the peaks (signals) have a value of 1 and the rest are zeros. The plot has global scaling between 0 and 1, and the black shades represent values above 0.5.

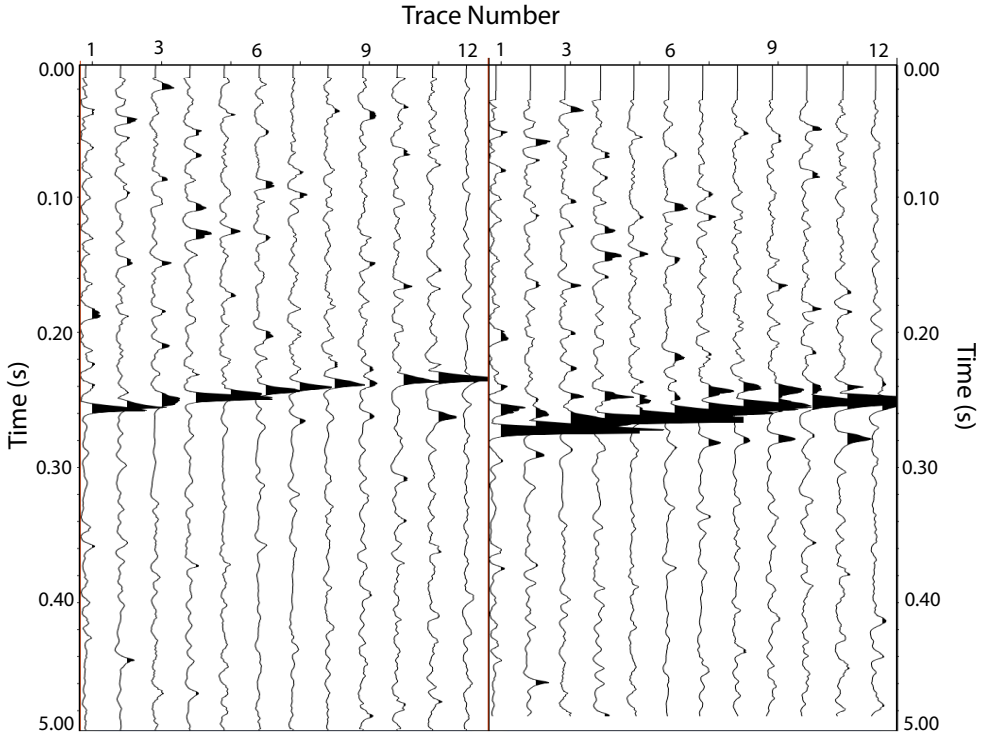


Figure 4.2: Left panel: shows the result of calculating the ratios of the average unweighted linearity between two moving windows with sizes of 7 ms. Right panel: shows the result of calculating the ratios of the average weighted linearity between two moving windows with sizes of 7 ms. The plots have global scaling between 0 and 4. The black shades represent peaks with ratios above 2.5.

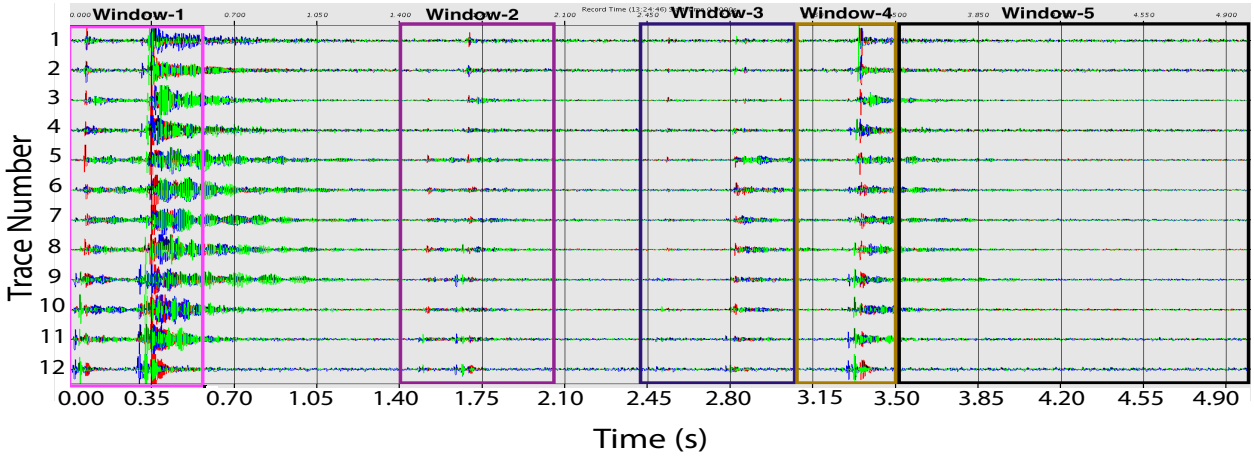
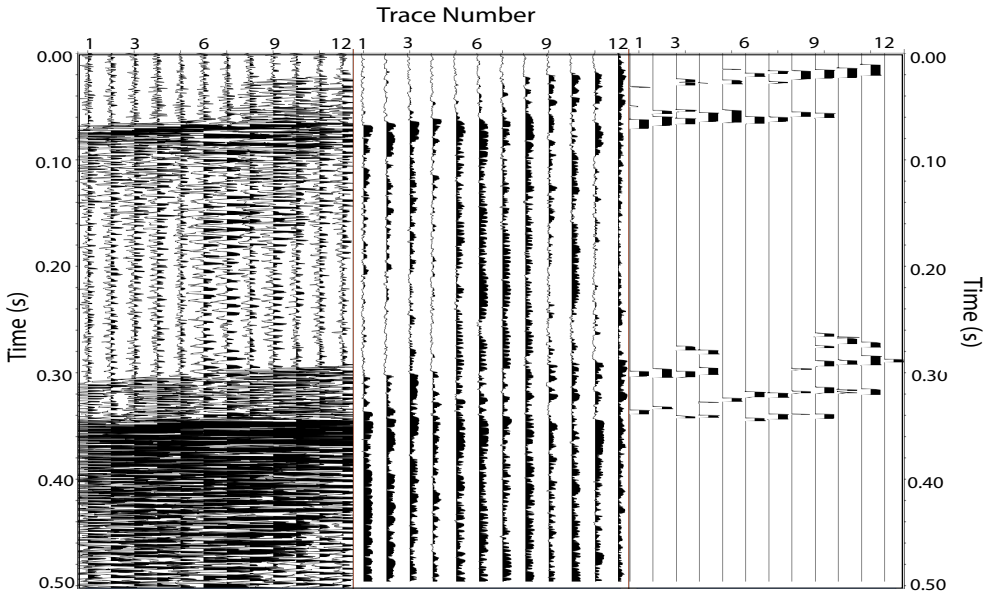
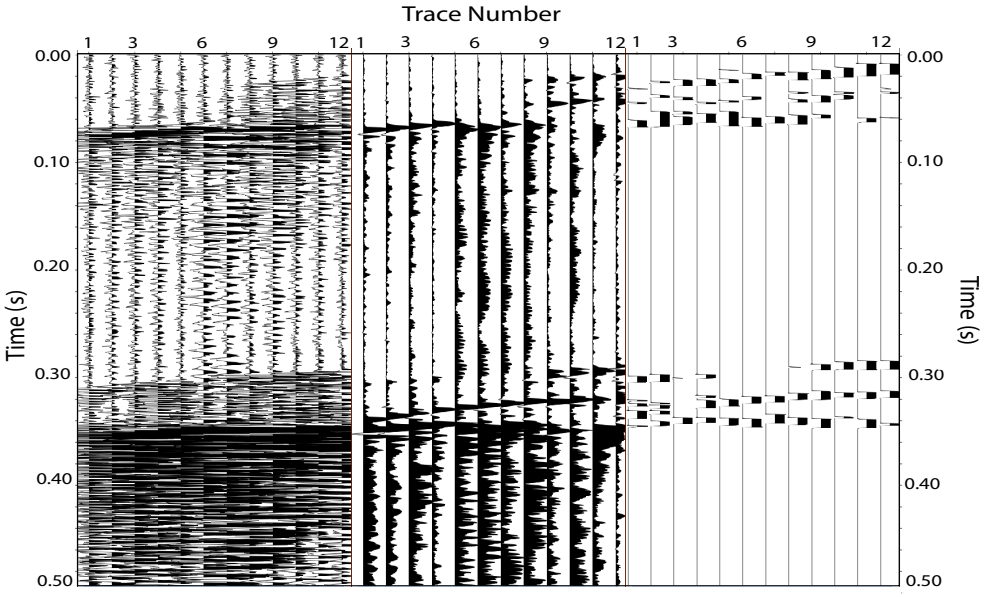


Figure 4.3: Shows time windows used in further analysis. Data is a fracturing file from stage 1. The data was filtered using a bandpass filter with frequencies of 5 Hz, 10 Hz, 250 Hz, and 300 Hz. The red waveform represents the vertical component (Z), the blue waveform represents the X component, and the green waveform represents the Y component.



(a)



(b)

Figure 4.4: Left panels: shows the vertical component of unfiltered data from window 1 Figure 4.3. Figure C.2 in appendix C shows the three components of this event. Middle panels: a) shows the result of the unweighted linearity calculation, and the plot has global scaling between 0 and 1. b) shows the result of the weighted linearity calculation. The plot has global scaling between 0 and 0.3. Right panels: show the result of detecting signal from the middle panels, where the peaks (signals) have a value of 1 and the rest are zeros. The plot has global scaling between 0 and 1. The black shades represent peak with values higher than 0.5.

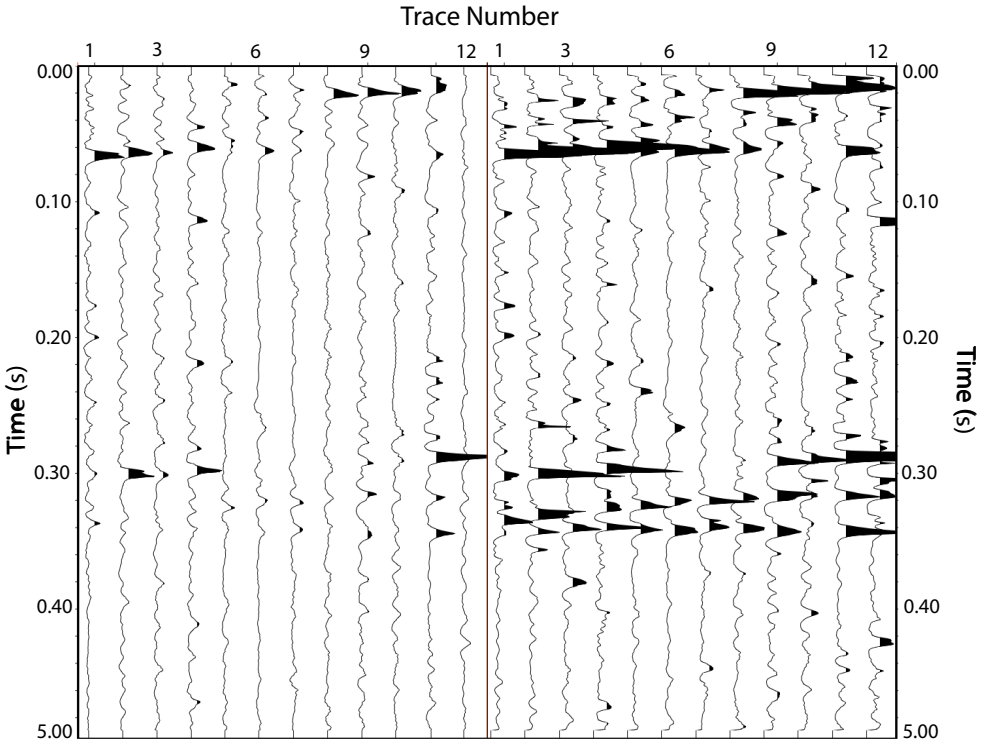
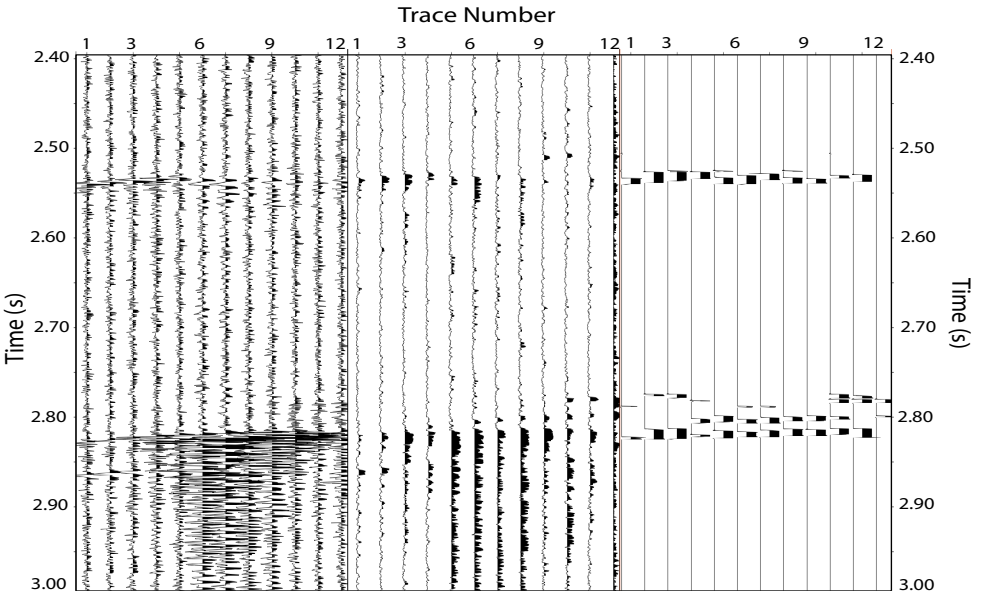
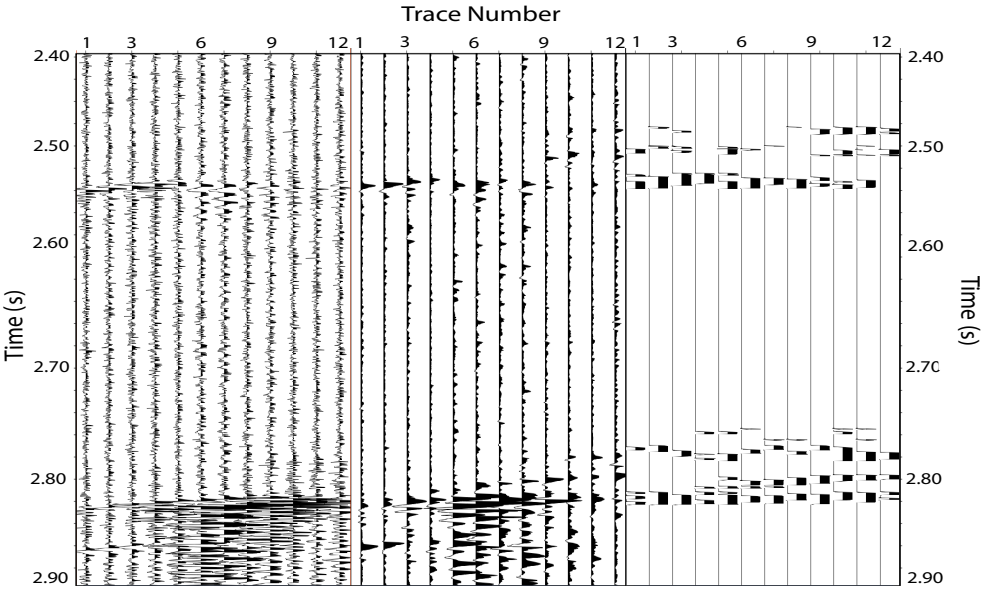


Figure 4.5: Left panel: shows the result of calculating the ratios of the average unweighted linearity in window 1 between two moving windows with sizes of 7 ms. Right panel: shows the result of calculating the ratios of the average weighted linearity in window 1 between two moving windows with sizes of 7 ms. The plots have global scaling between 0 and 4. The black shades represent peaks with ratios above 2.5.



(a)



(b)

Figure 4.6: Left panels: shows the vertical component of unfiltered data from window 3 Figure 4.3. Figure C.7 in appendix C shows the three components of this event. Middle panels: a) shows the result of the unweighted linearity calculation, and the plot has global scaling between 0 and 1. b) shows the result of the weighted linearity calculation. The plot has global scaling between 0 and 0.3. Right panels: show the results of detecting signal from the middle panels, where the peaks (signals) have a value of 1 and the rest are zeros. The plots have global scaling between 0 and 1. The black shades represent peak with values higher than 0.5.

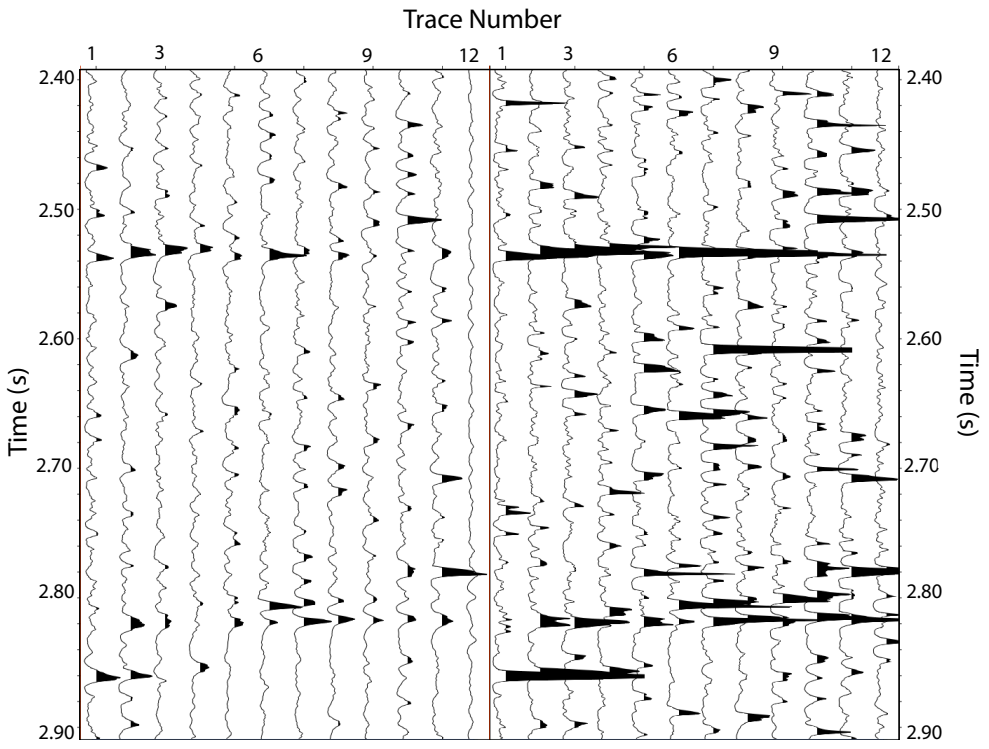


Figure 4.7: Left panel: shows the result of calculating the ratios of the average unweighted linearity in window 3 between two moving windows with sizes of 7 ms. Right panel: shows the result of calculating the ratios of the average weighted linearity in window 3 between two moving windows with sizes of 7 ms. The plots have global scaling between 0 and 4. The black shades represent peaks with ratios above 2.5.

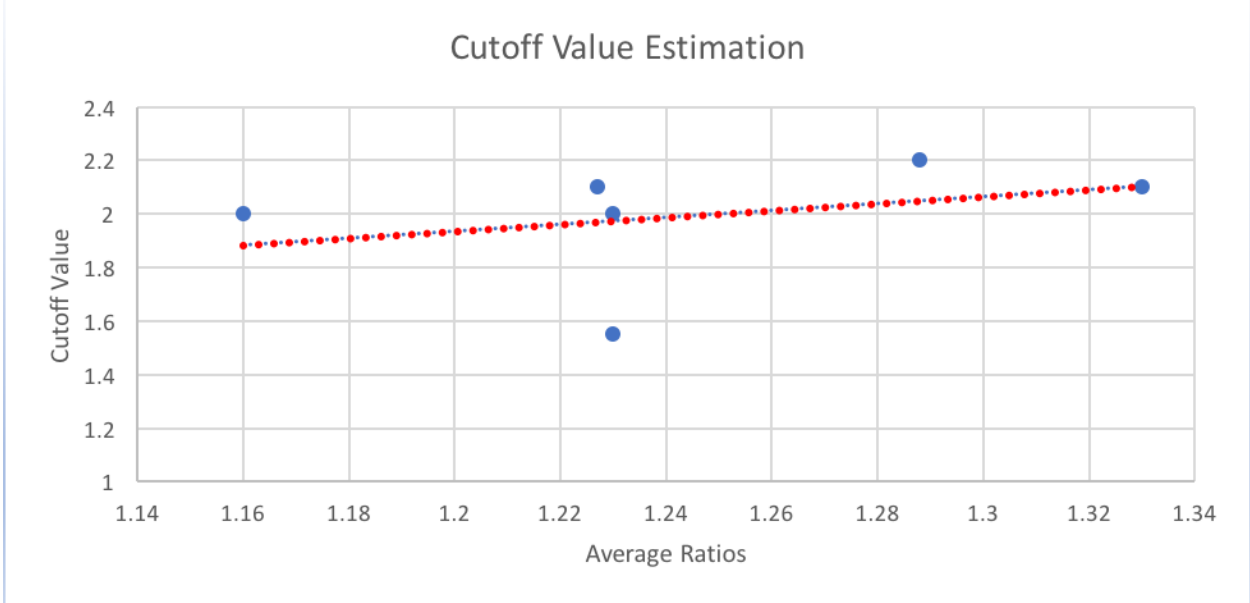


Figure 4.8: Shows the relationship between the cutoff value and the average ratios of weighted linearity. The blue dots represent the data from each of the 6 data files.

5. DISCUSSIONS

In this section I compare results from the unweighted linearity and the new, weighted linearity technique to show the improvements achieved by the addition of weighting.

5.1 Perforation Result

Comparing the unweighted linearity calculation (middle panel, Figure 4.1(a)) with the weighted linearity result (middle panel, Figure 4.1(b)) for the perforation event, it seems that both methods shows the P wave clearly with high linearity value. During the time of S arrival, the linearity result shows a broad window in time with high linearity values, with no distinct indication of the S arrival. This is because the P coda, the S wave, and the S coda are arriving at very close time, as it was verified by the hodogram results, that showed different linear behavior within the broad window. These behaviors suggest that signals are arriving in this window. Although the amplitude of the S wave is much larger than the P coda, both gave high linearity values since the linearity is not dependent on amplitudes. The new weighted linearity result show clear high peaks of linearity at the S arrival due to the incorporation of the displacement information of the S wave.

When comparing the final result of the signal detection techniques for both methods (right panels, Figures 4.1(a) and 4.1(b)), three arrivals (see Figure 5.1(a)) are more effectively identified in the new, weighted linearity method compared with only one signal, contaminated with noise, in the unweighted linearity method used in the past. The P arrival is identifiable in both methods, but is more contaminated with false signals (before the P arrival) in the unweighted linearity results. These false peaks are related to noise. The S wave arrival is fully detected in the weighted linearity method, while it is difficult to detect in the unweighted linearity method (there is only a small indication of S arrival on traces 7, 8, and 9). The S arrival is also shown in the ratios plot (Figure 4.2). The new, weighted linearity technique also shows a signal after the S wave, which is not shown in the unweighted linearity calculation and hard to see in the original data. This signal (S coda) is also indicated in the ratios plot (right panel, Figure 4.2).

5.2 Fracturing Results

5.2.1 Window 1 (0 s - 0.5 s)

For the early event (the medium strength event), the arrivals are clearer in the weighted linearity (middle panel, Figure 4.4(b)) compared to the original data (left panel, Figure 4.4(b)) and the unweighted linearity result (middle panel, Figure 4.4(a)). In addition, it has a good indication of the S arrivals (higher linearity values compared to the nearby signals) and fair indication of the P arrival. The S wave in the unweighted linearity result is somewhat identifiable but must be compared with the original data due to the P coda arrival, which arrives before the S arrival with the same linearity amplitude. Additionally, the P wave is poorly noticeable. For the later event (the strong event), both linearity methods showed clear P and S arrivals, however, it is much easier to identify them in the weighted linearity method.

Comparing the final results between the weighted linearity approach (right panel, Figure 4.4(b)) and the unweighted linearity approach (right panel, Figure 4.4(a)), it seems that both methods detect signals. However, the new method detects more signals and produces fewer false events. For example, the P wave at the medium event (top event) was detected clearly for all 12 traces in the weighted linearity results (right panel - Figure 4.4(b), Figure 5.1(b)), but was not detected in some of the traces in the unweighted linearity result (e.g. trace 2) and contaminated on others (e.g. trace 5 and 7) (right panel, Figure 4.4(a)). In addition, the P reflection (P coda) in the same event was fully detected in the weighted linearity approach, while the unweighted linearity result could not detect it, and this is also shown in the ratio results (Figure 4.5).

In window 1, I have clearly detected 7 signals using the new approach and only 5 signals contaminated with noise using the linearity approach (2 from the weak event and 3 from strong event), as shown in Figure 5.1(b). Both events (strong and medium) showed similar arrivals that are not noticeable in neither unweighted linearity nor the recorded data

5.2.2 Window 3 (2.4 s - 3 s)

The first event in the left panel (Figure 4.6(a)) at 2.5 s is one of the barely noticeable event, where the S wave is very hard to see in the filtered data and impossible to be noticed in the unfiltered data, but both linearity methods detected this arrival clearly. The P wave is fairly clear in the new, weighted linearity method, but is not in the unweighted method. The final result for the signal detect code for both techniques are shown in the right panels (Figures 4.6(a) and 4.6(b)). It seems that the weighted linearity (right panel, Figure 4.6(b)) detected the P, S and the S coda for this event (the early event or the barely noticeable event), as shown in Figure 5.1(c). The final result of the unweighted linearity (right panel Figure 4.6(a)) shows the S coda without any indication of P and S wave. In the ratios plot (Figure 4.7), the three signals are traceable in the weighted linearity ratios (right panel, Figure 4.7), and only one traceable signal (S coda) is shown in the unweighted linearity ratios (left panel, Figure 4.7).

The second event at 2.8 s in the left panel (Figure 4.6(a)), which is one of the weak events, has a clear S wave in the data, but not a clear P wave. Both techniques (middle panels, Figure 4.6(a) and 4.6(b)) show the S and S coda waves clearly. In addition, the final result of signal detect code (right panels, Figures 4.6(a) and 4.6(b)) also show these two signals. The P wave for this weak event (later event in window 3) is fairly detectable in weighted linearity and undetectable in the unweighted linearity. In the final result of the unweighted linearity (right panels, Figure 4.6(a)), the P arrival is only shown on five traces (traces 6,7, 10, 11, and 12). However, it appears in most of the traces, with some noise, in the result of the new, weighted linearity method shown in Figure 4.6(b) (right panel).

In this window, I have detected 6 signals, split between two events, using the new method (See Figure 5.1(c)) and only three signals using the traditional, unweighted linearity. These 6 signals are not detectable in original data.

To make sure that my new, weighted linearity technique is not generating any fake events, I chose a file from the early section of stage 1 which only contains noise, as shown in Figure 5.2. The new weighted linearity method was also applied to the first 0.5 s of the file, and the result is

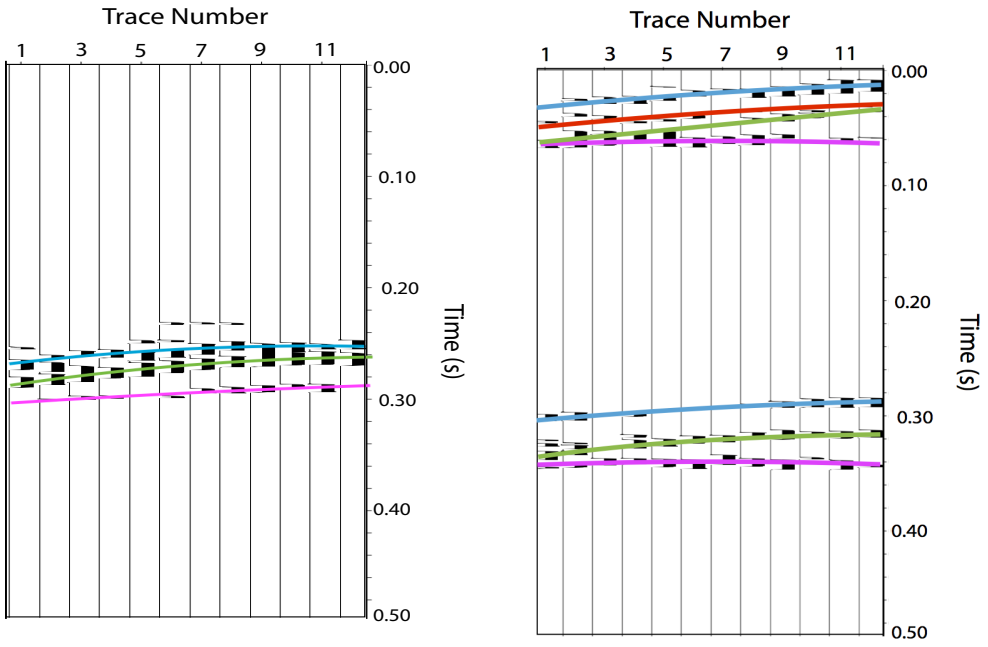
shown in Figure 5.3 along with the final result of signal detect. In both of these tests, no false events were generated from noise using the weighted linearity technique.

5.3 General Discussion

In the tests presented here, my new approach detected 16 signals from the two examples file, while the traditional, unweighted linearity only detected 9 signals. This indicates that there is an average of 40% improvement in signal detection using the new approach. In addition, most of the signals are not clearly detectable in the original data that can be detected using the new method. In the case where both the weighted and unweighted linearity methods detected the same number of signals, the weighted linearity showed more detail in the signal and less noise compared to unweighted linearity results.

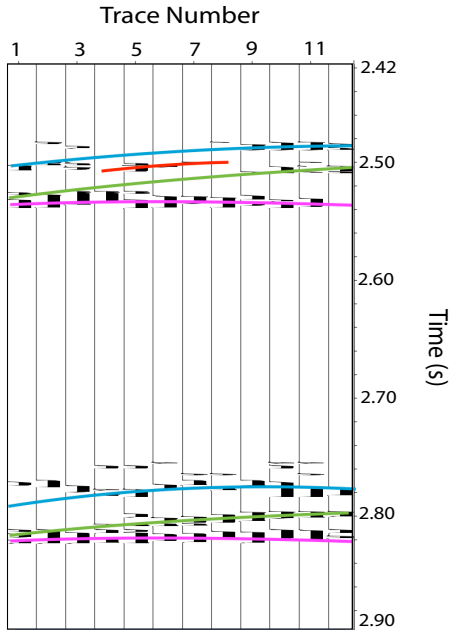
In some cases (mainly the long files in time), the weighted linearity shows a few noise peaks, especially before the P arrival, due to the cutoff value that is applied for different strength events within a single file. To overcome this issue, it is better separate the targeted events into different files and then apply the signal detection code to each separately.

The ratio plots for the new weighted linearity method have higher ratios by an average of 70% compared to the ratios for the conventional linearity. Additionally, the patterns (relative arrival times) of the detected signals in the weighted linearity schemes are very similar for all events in the fracturing data. This similarity is fairly clear in the processed results, but it is not at all evident in the original data. Because the microseismic events (fracture slip) occurred with a few seconds during the fracture stage, it is likely that they correspond to similar source mechanisms, which would be expected to generate similar P and S waves.



(a)

(b)



(c)

Figure 5.1: Show the signal detection result from the new weighted linearity method of a) the perforation example (right panel, Figure 4.1(b)), b) window 1 (right panel, Figure 4.4(b)), and c) window 3 (right panel, Figure 4.6(b)). The blue, red, green and magenta curves represent the P, P coda, S, and S coda waves, respectively.

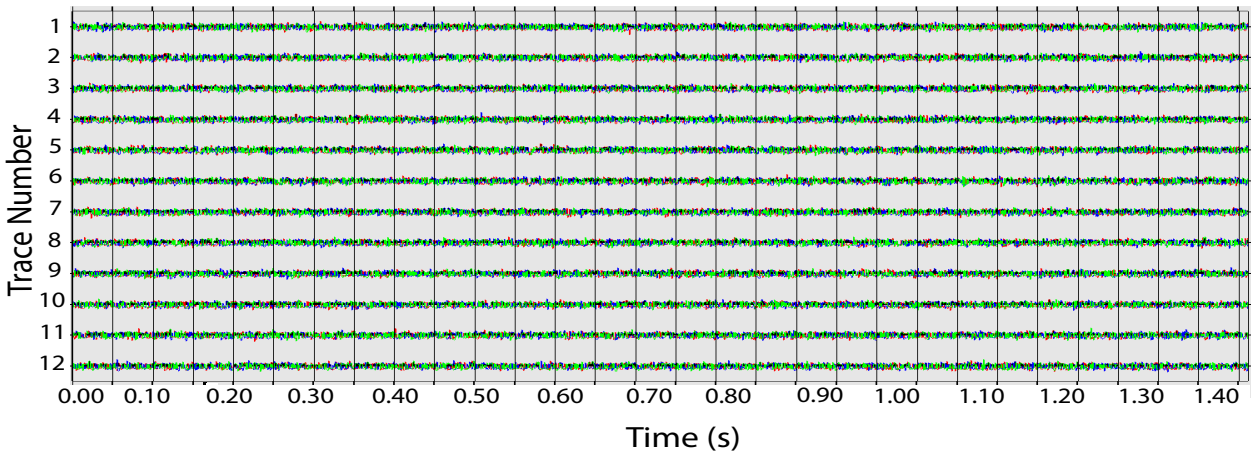


Figure 5.2: Shows a fracturing file from early stage 1. The red waveform represents the vertical component (Z), the blue waveform represents the X component, and the green waveform represents the Y component.

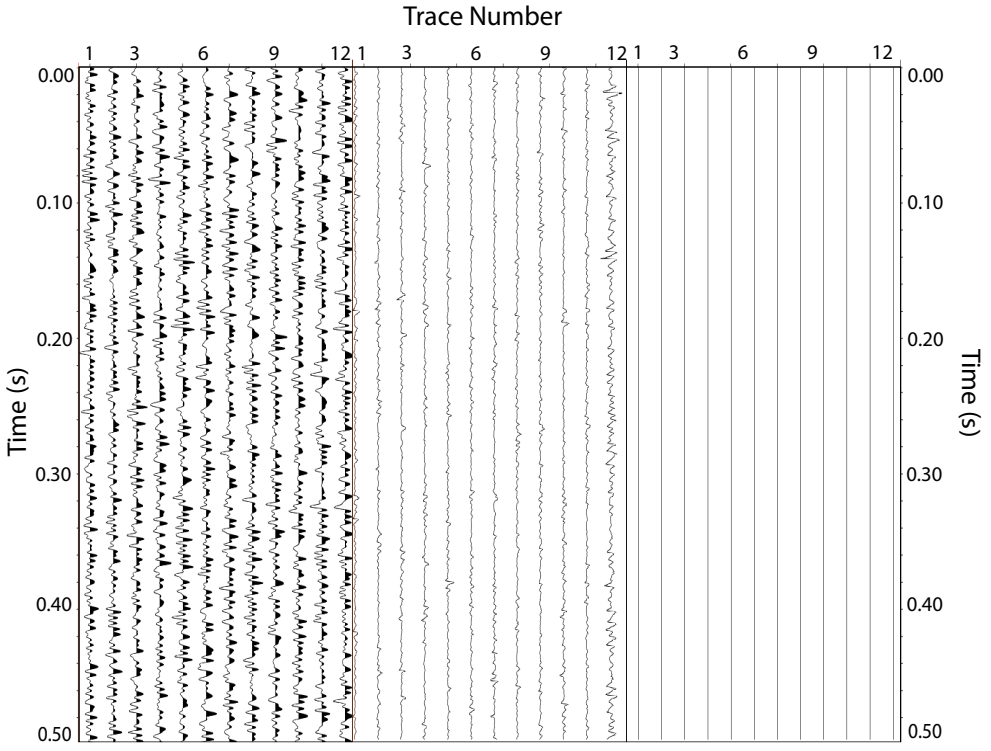


Figure 5.3: Left panel: shows the vertical component of the unfiltered data in Figure 5.2. Middle panel: shows the result of the weighted linearity method. The plot has global scaling between 0 and 0.3. Right panel: shows the result of detecting signal from the new, weighted linearity approach (middle panel). The plot has global scaling between 0 and 1.

6. CONCLUSION

Signal detection is essential for microseismic monitoring. There are many techniques that detect signals, with one notable technique using the linearity of particle motion. Linearity is amplitude independent, so in some cases the signals are not as clear in the linearity results compared to the original data. By taking into account the amplitude of displacement I have improved the standard linearity method. The displacement was optimally measured using the envelope of a signal. The new method (weighted linearity) multiplies a linearity calculation with displacement from the envelope, and the results show a 40% improvement in number of arrivals detected, detecting signals strengths ranging from barely noticeable to strong. In addition, the weighted linearity method shows a few noise peaks and more detail of signals compared to the traditional, unweighted linearity method. This improvement has multiple applications, such as improving estimates in microseismic event location by having more accurate arrival times and identifying reflections and other signals that cannot be distinguished with the traditional linearity calculation. Therefore, this new method will be helpful in extracting additional information from microseismic data beyond traditional location and source studies.

REFERENCES

[1] D. Billard, *Map and cross sectional view of the project setup [figure]*. Halliburton, feb 2017.

[2] Y. Aoki, "Monitoring temporal changes of seismic properties," *Frontiers in Earth Science*, vol. 3, p. 42, jul 2015.

[3] Y. Li, H. Wang, M. Fehler, and Y. Fu, "Wavefield characterization of perforation shot signals in a shale gas reservoir," *Physics of the Earth and Planetary Interiors*, vol. 267, pp. 31–40, jun 2017.

[4] A. Hakso, M. D. Zoback, J. Du, A. Hakso, and M. D. Zoback, "Utilization of Microseismic Multiplets to Detect Velocity Changes During Multi-Stage Hydraulic Fracturing in the Barnett Shale," in *SEG Technical Program Expanded Abstracts 2015*, pp. 2573–2577, Society of Exploration Geophysicists, aug 2015.

[5] L. Li, D. Becker, H. Chen, X. Wang, and D. Gajewski, "A systematic analysis of correlation-based seismic location methods," *Geophysical Journal International*, vol. 212, pp. 659–678, jan 2018.

[6] Y. Mukuhira, M. Fehler, and M. Hirokazu, "Buried microseismic detection from continuous data using spectral matrix analysis of triaxial particle motion," in *SEG Technical Program Expanded Abstracts 2017*, pp. 2797–2801, Society of Exploration Geophysicists, aug 2017.

[7] Z. Zhang, J. W. Rector, and M. J. Nava, "Simultaneous inversion of multiple microseismic data for event locations and velocity model with Bayesian inference," *GEOPHYSICS*, vol. 82, pp. KS27–KS39, may 2017.

[8] B. Witten and J. Shragge, "Microseismic image-domain velocity inversion: Marcellus Shale case study," *GEOPHYSICS*, vol. 82, pp. KS99–KS112, nov 2017.

[9] D. Price, D. Angus, K. Chambers, and G. Jones, "Surface microseismic imaging in the presence of high-velocity lithologic layers," *GEOPHYSICS*, vol. 80, pp. WC117–WC131, nov

2015.

[10] J. Li, C. Li, S. A. Morton, T. Dohmen, K. Katahara, and M. Nafi Toksöz, "Microseismic joint location and anisotropic velocity inversion for hydraulic fracturing in a tight Bakken reservoir," *GEOPHYSICS*, vol. 79, pp. C111–C122, sep 2014.

[11] H.-I. Choi and W. Williams, "Improved time-frequency representation of multicomponent signals using exponential kernels," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 37, pp. 862–871, jun 1989.

[12] J. Akram and D. W. Eaton, "A review and appraisal of arrival-time picking methods for downhole microseismic data," *GEOPHYSICS*, vol. 81, pp. KS71–KS91, mar 2016.

[13] M. Withers, R. Aster, C. Young, J. Beiriger, M. Harris, S. Moore, and J. Trujillo, "A Comparison of Select Trigger Algorithms for Automated Global Seismic Phase and Event Detection," *Bulletin of the Seismological Society of America*, vol. 88, no. 1, pp. 95–106, 1998.

[14] J. C. Samson, "Matrix and Stokes vector representations of detectors for polarized waveforms: theory, with some applications to teleseismic waves," *Geophysical Journal International*, vol. 51, pp. 583–603, dec 1977.

[15] H. Moriya, K. Nagano, and H. Niitsuma, "Precise source location of AE doublets by spectral matrix analysis of triaxial hodogram," *GEOPHYSICS*, vol. 59, pp. 36–45, jan 1994.

[16] N. Soma and H. Niitsuma, "Identification of structures within the deep geothermal reservoir of the Kakkonda field, Japan, by a reflection method using acoustic emission as a wave source," *Geothermics*, vol. 26, pp. 43–64, feb 1997.

[17] N. Soma, H. Niitsuma, and R. Baria, "Reflection technique in time-frequency domain using multicomponent acoustic emission signals and application to geothermal reservoirs," *GEOPHYSICS*, vol. 67, pp. 928–938, may 2002.

[18] K. Nagano, H. Niitsuma, and N. Chubachi, "Automatic algorithm for triaxial hodogram source location in downhole acoustic emission measurement," *GEOPHYSICS*, vol. 54, pp. 508–513, apr 1989.

[19] M. J. Mayerhofer, N. A. Stegent, J. O. Barth, and K. M. Ryan, "Integrating Fracture Diagnostics and Engineering Data in the Marcellus Shale," in *SPE Annual Technical Conference and Exhibition*, Society of Petroleum Engineers, apr 2011.

[20] H. Moriya and H. Niitsuma, "Precise detection of a P-wave in low S/N signal by using time-frequency representations of a triaxial hodogram," *GEOPHYSICS*, vol. 61, pp. 1453–1466, sep 1996.

[21] C. O. Kanu, R. Snieder, and D. O 'connell, "Estimation of velocity change using repeating earthquakes with different locations and focal mechanisms," *JOURNAL OF GEOPHYSICAL RESEARCH: SOLID EARTH J. Geophys. Res. Solid Earth*, vol. 11850206, no. 118, pp. 2905–2914, 2013.

[22] C. O. Kanu and A. Muñoz, "Extraction of velocity changes from repeating microseismic events via dynamic time warping," *Center of Wave Phenomena Reports*, 2013.

[23] D. J. Robinson, M. Sambridge, and R. Snieder, "A probabilistic approach for estimating the separation between a pair of earthquakes directly from their coda waves," *JOURNAL OF GEOPHYSICAL RESEARCH: SOLID EARTH J. Geophys. Res. Solid Earth*, vol. 116, 2011.

[24] E. Alexander and D. Poularikas, *The Handbook of Formulas and Tables for Signal Processing.* Boca Raton, Florida 33431.: CRC PRESS, 1999.

APPENDIX A

COMPARISON BETWEEN THE FOUR APPROACHES OF DISPLACEMENT MEASUREMENTS

In this section I will compare the four different measurements of displacement after multiplying them with the unweighted linearity result from stage 1 fracturing data and stage 2 perforation data.

The first approach calculates the square root of the average of squared amplitude within a window (equation 3.8). The second approach calculates the average amplitudes (equation 3.9), and will be more sensitive to the relatively low amplitudes. The third approach uses the square root of the second approach (equation 3.10), decreasing the effect of displacement and increasing the effect of the linearity. The fourth approach uses the square root of the envelope of a signal (equation 3.11).

Panel (1) in Figure A.1 shows the bandpass filtered signal (frequencies: 0 Hz, 10 Hz, 250 Hz, and 300 Hz) from stage 2 perforation data, where the P and S arrival are clear. Panel (2) is the resulted envelope from panel (1). Panel (3) shows the unweighted linearity result calculated from panel (1). Panel (4) is the result of multiplying panel (3) with the first approach (equation 3.8). Panels (5, 6, and 7) are the results from multiplying panel (3) (unweighted linearity) with equations (3.9,3.10, and 3.11) accordingly.

In general, it seems that all four approaches (panels 4 through panel 7, Figure A.1) show better identification of P and S arrival compared to the unweighted linearity result (panel 3). Furthermore, approaches three and four (panel 6 and 7) show more sensitivity toward the low amplitudes of P waves compared the to approaches one and two. Approaches three and four also show more of the P coda (peaks in between P and S arrivals) compared to the approaches one and two. I have performed the same test on multiple perforation files and fracturing files (stages 2, 4 and 7), and I concluded that the envelope approach (fourth approach, equation 3.11) and the square root of average amplitudes (third approach, equation 3.10) always show better arrival identification than

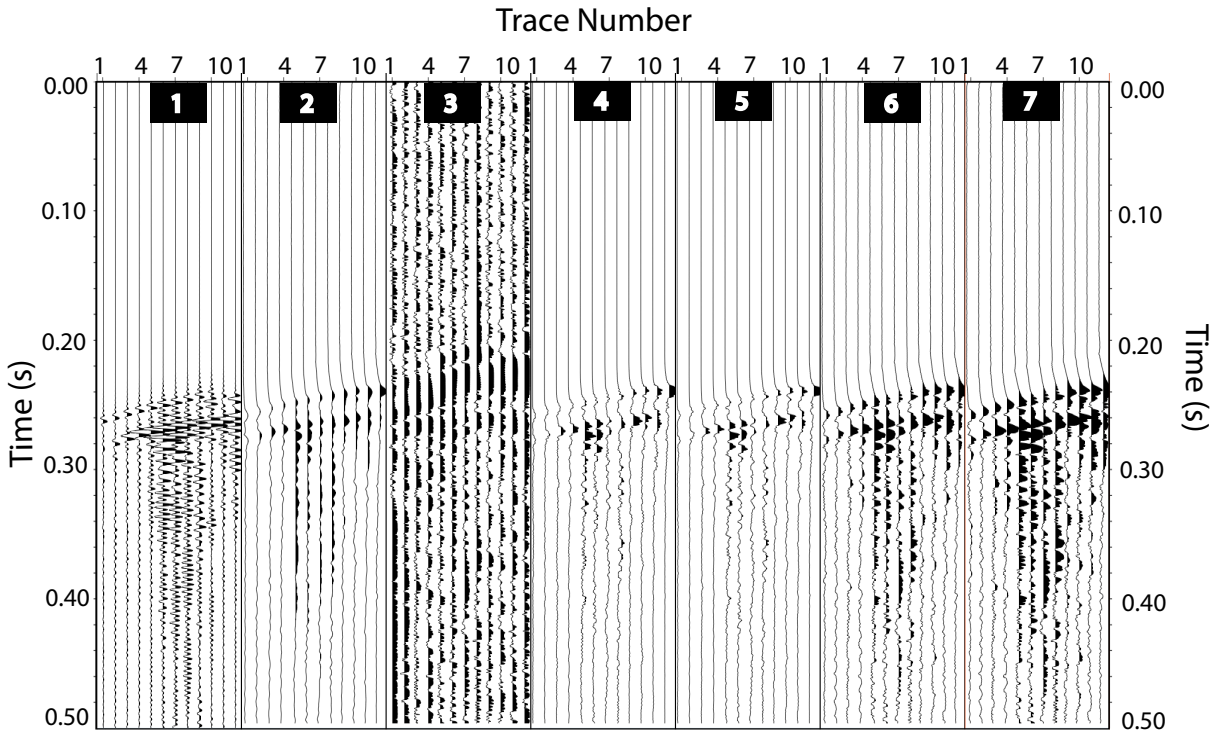


Figure A.1: Panel 1: shows the vertical component of data from a perforation event in stage 2. The data was filtered using a bandpass filter with frequencies of 0 Hz, 10 Hz, 250 Hz, and 300 Hz. Panel 2: shows the resulted envelope from panel (1). Panel 3: shows the unweighted linearity calculation from panel (1). The plot has global scaling between 0 and 1. Panels 4: displays the result of the first approach, which multiplies Panel (3) by equation (3.8). Panels 5: displays the result of the second approach, which multiplies Panel (3) by equation (3.9). Panels 6: displays the result of the third approach, which multiplies Panel (3) by equation (3.10). Panels 7: displays the result of the fourth approach, which multiplies Panel (3) by equation (3.11). The plots (4 through 7) have global scaling between 0 and 0.4.

the first two approaches.

In Figure A.1, there are not significant differences between approach three (panel 6) and approach four (panel 7). However, in the fracturing data example (stage 1) (see Figure A.2), the envelope result (approach four; Figure A.2 (panel 5)) shows better P wave, and P coda identification compared to the square root of average amplitudes (third approach, Figure A.2 (panel 4)). Similar results were shown in the rest of the fracturing stages and later stages in the perforation data. Therefore, I can conclude that, compared to the other approaches, approach 4 (the envelope of a signal) is the optimal method to measure displacement and multiply with linearity to get the

new weighted linearity result.

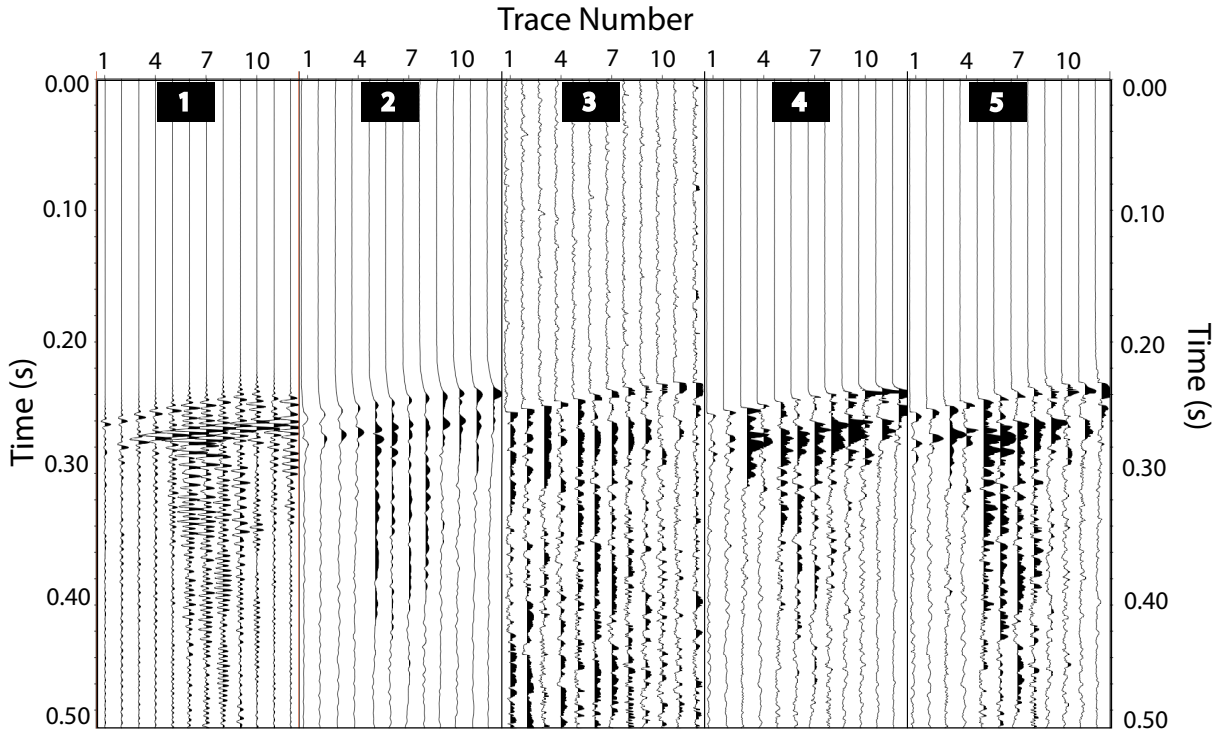


Figure A.2: Panel 1: shows the vertical component of data from a fracturing event in stage 1. The data was filtered using a bandpass filter with frequencies of 0 Hz, 10 Hz, 250 Hz, and 300 Hz. Panel 2: shows the resulted envelope from panel (1). Panel 3: shows the unweighted linearity calculation from panel (1). The plot has global scaling between 0 and 1. Panels 4, and 5: shows the result from multiplying panel (3) with the third approach and fourth approach, respectively. The plots have global scaling between 0 and 0.4.

APPENDIX B

HILBERT TRANSFORM IN FREQUENCY DOMAIN

The Hilbert transform of a cosine function is a sine function in the time domain [Alexander and Poularikas, 1999]. The same transformation can be done in the frequency domain by multiplying the frequencies of the cosine function and sine function with imaginary numbers i and -i:

$$Y(f) * i \qquad for \;\; f \geq \frac{N}{2} \tag{B.1}$$

$$Y(f) * -i \qquad for \;\; f \leq \frac{N}{2} \tag{B.2}$$

I can show how this work by applying it to frequencies corresponding to a function $cos(t)$. Suppose I have a Fourier transform defined on a domain ($-N/2$ to $N/2$), which is equivalent to a Fourier transform on a domain $(0, N)$, as shown in Figure B.1. Suppose the real variable (a) is the amplitude for the frequencies (1 Hz and -1 Hz). Applying the inverse Fourier transform to those two frequencies will give a $cos(t)$:

$$\begin{aligned} a * \cos(t) &= \frac{1}{2} * (y * (1) * e^{i*(1)*t} + y * (-1) * e^{i*(-1)*t}) \\ &= \frac{1}{2} * (a * e^{i*t} + a * e^{-i*t}) \\ &= \frac{1}{2} * a * [(\cos(t) + i\sin(t)) + (\cos(t) - i\sin(t))] \\ &= a * \cos(t). \end{aligned} \tag{B.3}$$

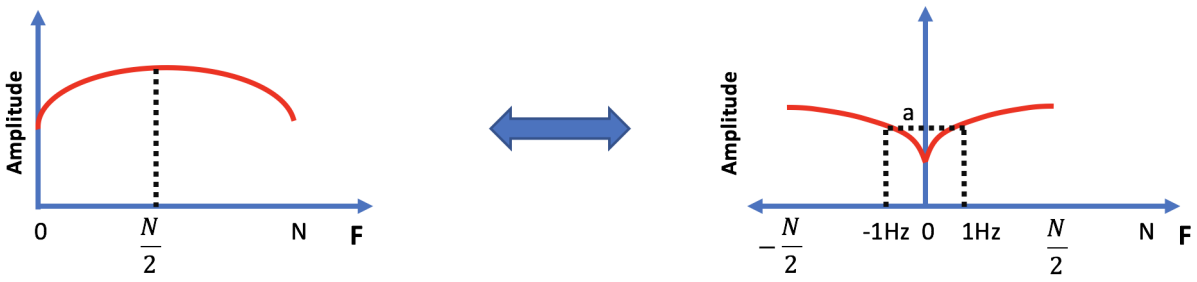


Figure B.1: Left panel: Shows a schematic example of a *cosine* function in the frequency domain that has a range of (0,N). Right panel: shows an equivalent display of the function in the left panel

If I multiply the frequencies with i and $-i$, the inverse Fourier transform will be a $sin(t)$:

$$\begin{aligned} a * \cos(t) &= \frac{1}{2} * (y * (1) * -i * e^{i*(1)*t} + y * (-1) * i * e^{i*(-1)*t}) \\ &= \frac{1}{2} * (a * -i * e^{i*t} + a * i * e^{-i*t}) \\ &= \frac{1}{2} * a * [-i * (\cos(t) + i\sin(t)) + i * (\cos(t) - i\sin(t))] \\ &= \frac{1}{2} * a * [-i * \cos(t) + \sin(t) + i * \cos(t) + \sin(t))] \\ &= \frac{1}{2} * a * 2 * \sin(t) \\ &= a * \sin(t). \end{aligned} \tag{B.4}$$

Therefore, multiplying by i and i, I have change the inverse Fourier transform for cosine to sine.

APPENDIX C

RESULTS AND DISCUSSIONS

In this section, the extra figures for the results will be presented. In addition, windows 2, 4, and 5 in Figure 4.3 are discussed.

C.1 Perforation Data

Figure C.1 shows the three components of the perforation event.

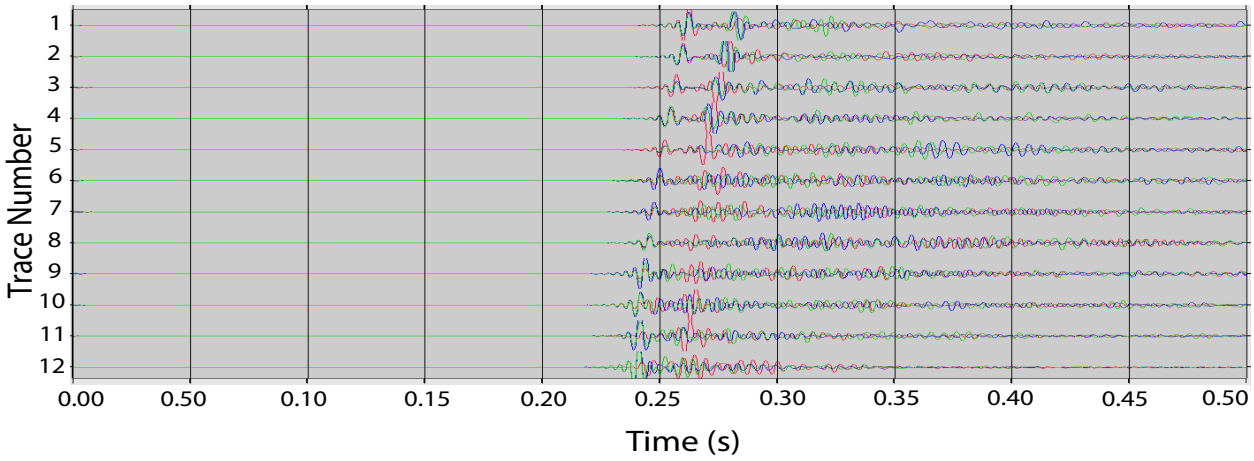


Figure C.1: shows a perforation file from stage 2. The event was filter by a bandpass filter with frequencies of 5 Hz, 10 Hz, 250 Hz, and 300 Hz. The red waveform represents the vertical component (Z), the blue waveform represents the X component, and the green waveform represents the Y component.

C.2 Fracturing Data

C.2.1 Window 1 (0 s - 0.5 s)

Figure C.2 shows the three components of the fracturing events in window 1.

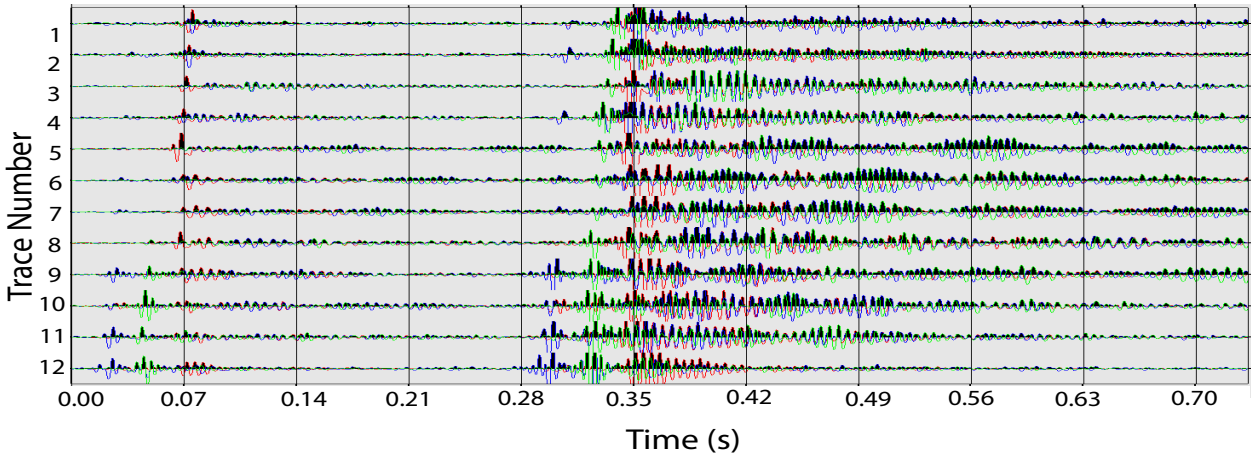


Figure C.2: shows a zoomed plot of Figure 4.3 at window 1. The different waveformsâĂŹ color represents the different component of a geophone, where red, blue and green are Z, X, and Y, respectively.

C.2.2 Window 2 (1.4 s - 2 s)

Figure C.3 shows a zoomed image of window 2, which includes a weak and barely noticeable events. The unweighted and weighted linearity result are shown in Figures C.4(a) and C.4(b) along with the final result of signal detection.

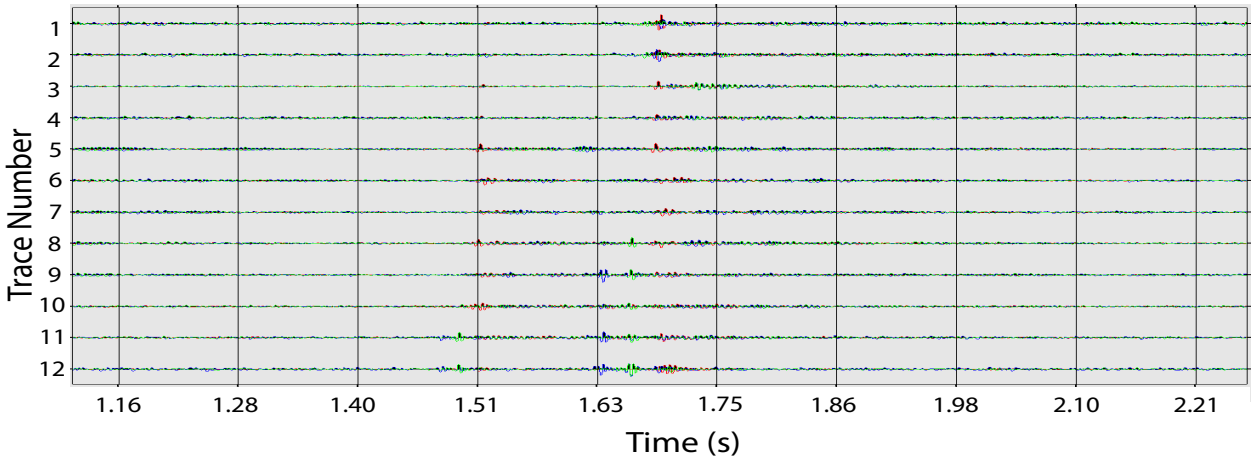
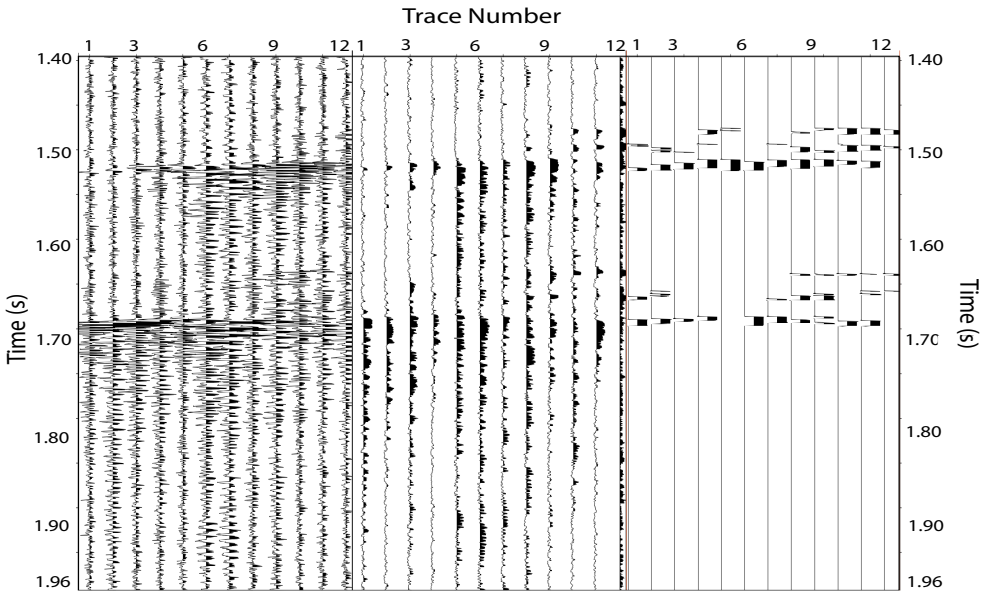
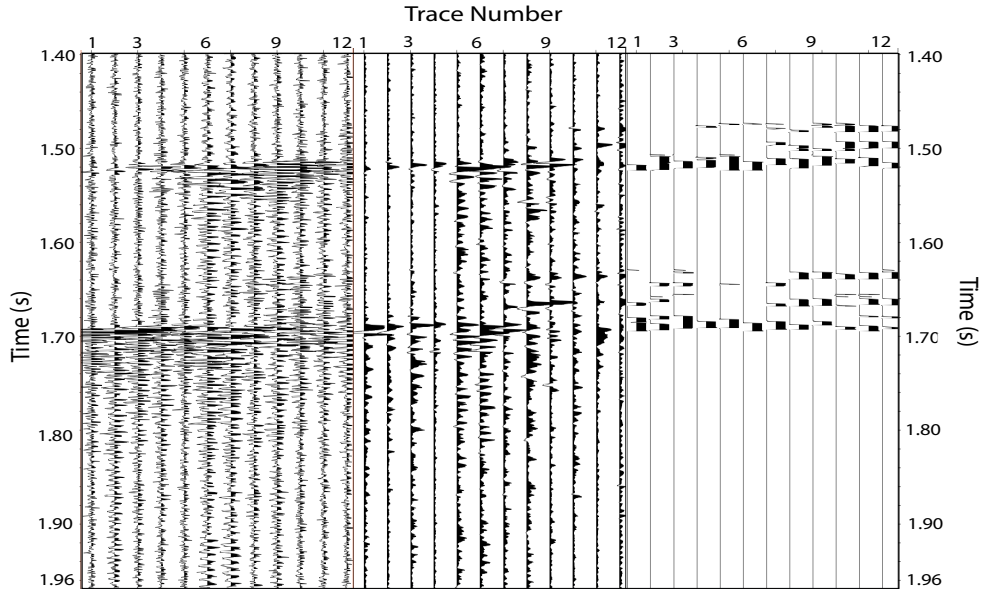


Figure C.3: Shows a zoomed plot of Figure 4.3 at window 2. The different waveforms' color represents the different component of a geophone, where red, blue and green are Z, X, and Y, respectively.



(a)



(b)

Figure C.4: Left panels: show the vertical component of unfiltered data from window 2 Figure 4.3. Middle panels: a) shows the result of the unweighted linearity calculation, and the plot has global scaling between 0 and 1. b) shows the result of the weighted linearity calculation. The plot has global scaling between 0 and 0.3. Right panels: show the result of detecting signal from the middle panels, where the peaks (signals) have a value of 1 and the rest are zeros. The plots have global scaling between 0 and 1. The black shades represent peak with values higher than 0.5.

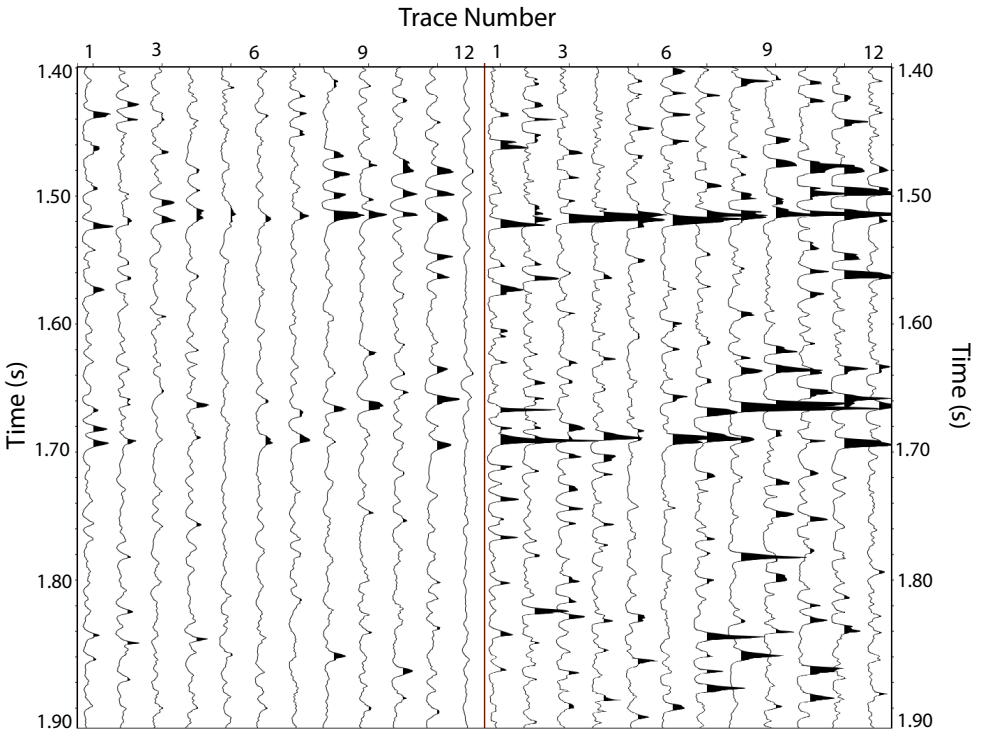


Figure C.5: Left panel: shows the result of calculating the ratios of the average unweighted linearity in window 2 between two moving windows with sizes of 7 ms. Right panel: shows the result of calculating the ratios of the average weighted linearity in window 2 between two moving windows with sizes of 7 ms. The plots have global scaling between 0 and 4. The black shades represent peaks with ratios above 2.5.

Window 2 contains two events (barely noticeable and weak), as shown in Figure C.3. In the weighted linearity result (middle panel, Figure C.4(b)), the S, S coda waves for the earlier event (barely noticeable event) are clearly detectable, but the P wave is hard to identify. Many parts of this P wave for this event are identified in the final result of signal detect (right panel, Figure C.4(b)) along with the S and S coda, as shown in Figure C.6. The S coda and some parts of S wave for the same event (barely noticeable event) are the only signals shown in the unweighted linearity result (middle panel, Figure C.4(a)). The final result for the unweighted linearity method (right panel, Figure C.4(a)) shows the three signals (P, S and S coda). For the later event (weak event), all three signals are shown clearly in the new, weighted linearity result (middle panel, Figure C.4(b)) and only two signals (S, and S coda) are shown in the traditional, unweighted linearity (middle

panel, Figure C.4(a)). The final results (right panels, Figure C.4(a) and C.4(b)) for the later event indicate that both method can detect the three signals; however, the new technique has more peaks that represent the signals than the unweighted linearity method. Furthermore, the ratios (Figure C.5) for the weighted linearity is about 70% larger and more identifiable compared to the ratios of the unweighted linearity method.

In window 2, both methods detected 6 signals (see Figure C.6) that cannot be seen in the original data, but the new technique was more detailed, showing a higher number of peaks.

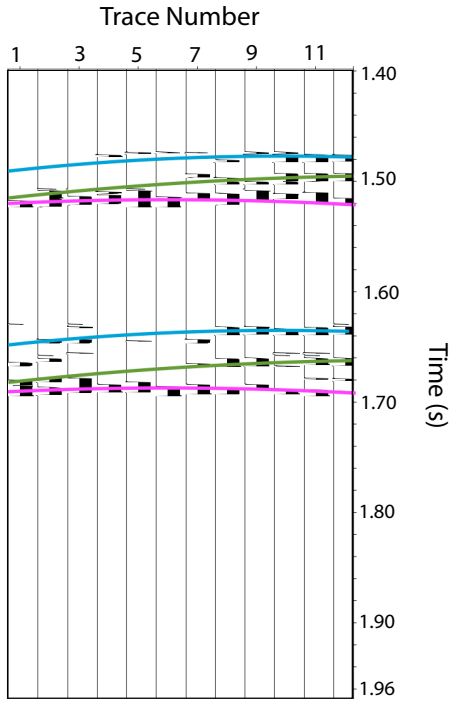


Figure C.6: Shows the signal detection result from the new weighted linearity method (middle panel, Figure C.4(b)). The blue, green, and magenta curves represent the P, S, and S coda waves, respectively.

C.2.3 Window 3 (2.4 s - 3 s)

Figure C.7 shows the three components of the perforation event in window 3.

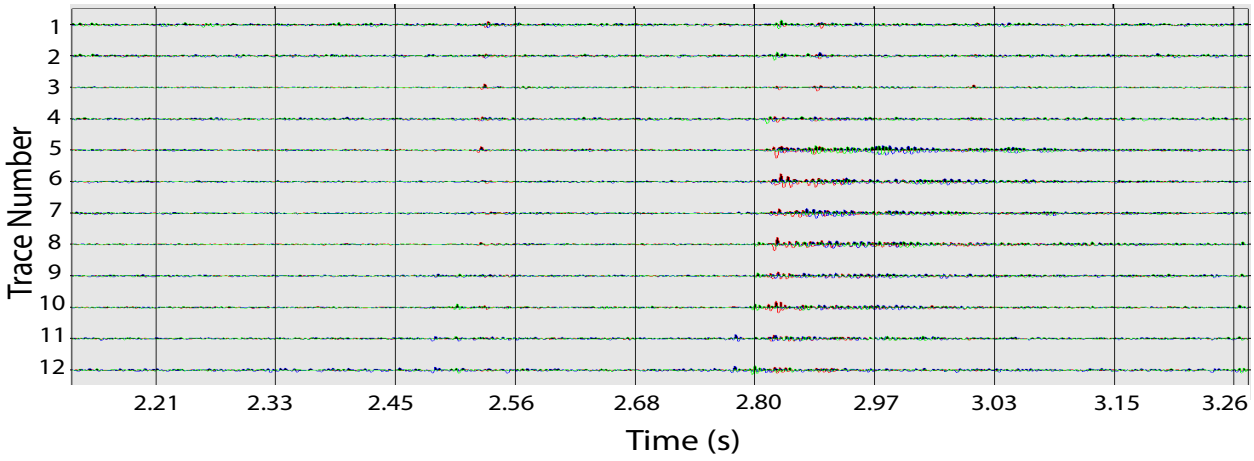


Figure C.7: shows a zoomed plot of Figure 4.3 at window 3. The different waveforms' color represents the different component of a geophone, where red, blue and green are Z, X, and Y, respectively.

C.2.4 Window 4 (3 s - 3.5 s)

Figure C.8 shows a zoomed image of window 4, which includes a medium and barely noticeable events. The unweighted and weighted linearity result are shown in Figures C.9(a) and C.9(b) along with the final result of signal detection.

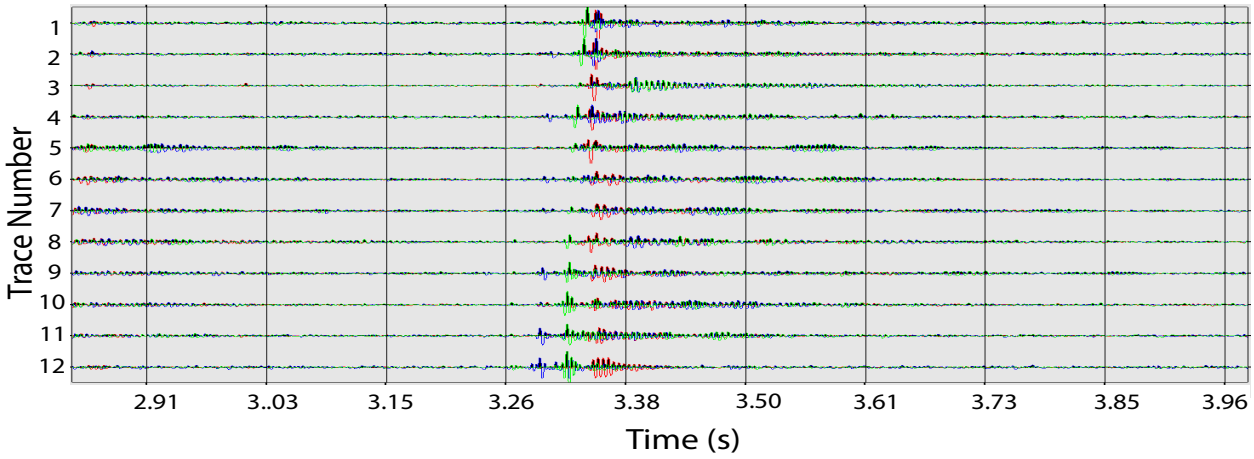
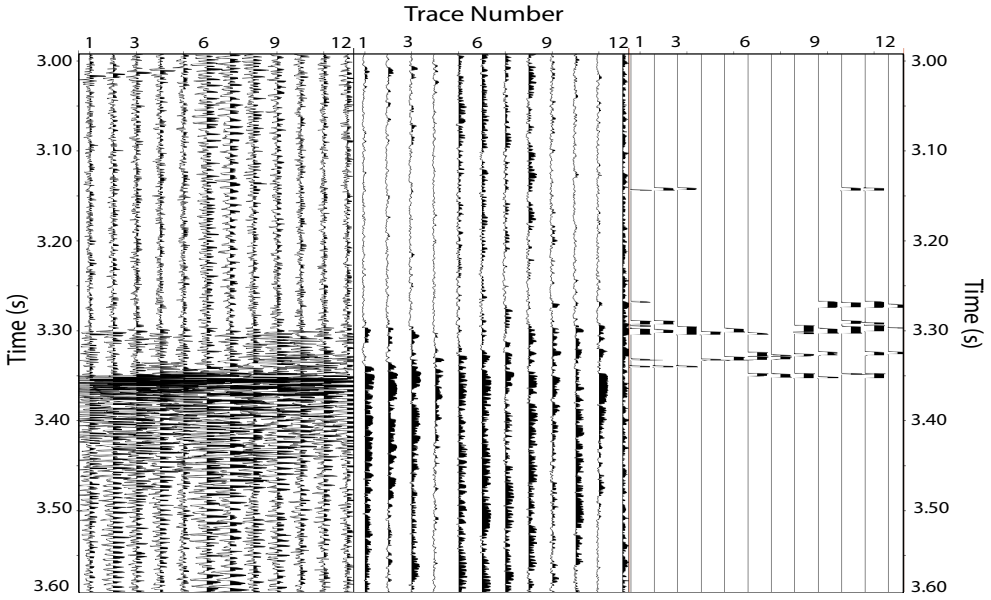
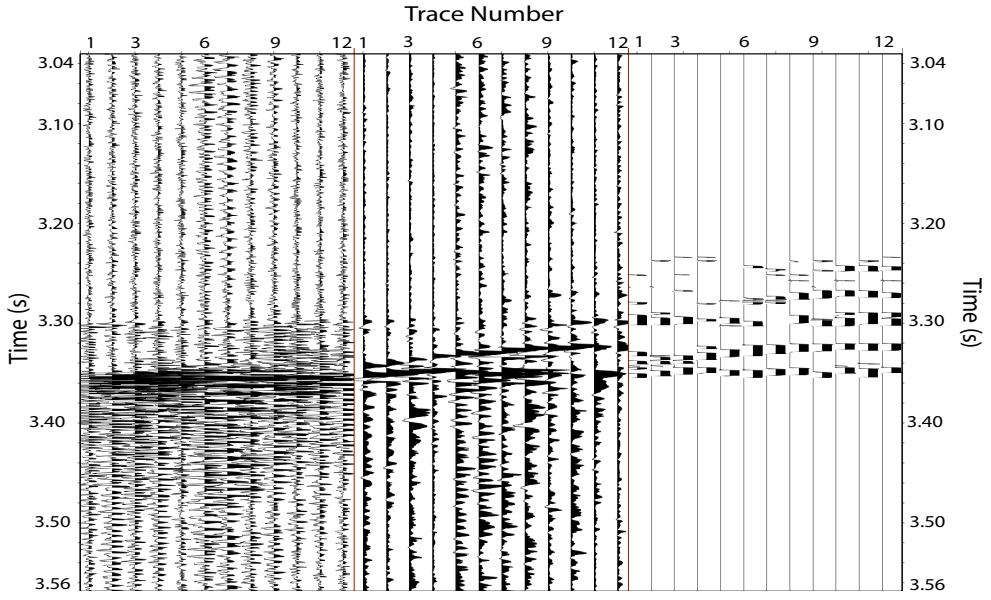


Figure C.8: Shows a zoomed plot of Figure 4.3 at window 4. The different waveforms' color represents the different component of a geophone, where red, blue and green are Z, X, and Y, respectively.



(a)



(b)

Figure C.9: Left panels: show the vertical component of unfiltered data from window 4 Figure 4.3. Middle panels: a) shows the result of the unweighted linearity calculation, and the plot has global scaling between 0 and 1. b) shows the result of the weighted linearity calculation. The plot has global scaling between 0 and 0.3. Right panels: show the result of detecting signal from the middle panels, where the peaks (signals) have a value of 1 and the rest are zeros. The plots have global scaling between 0 and 1. The black shades represent peak with values higher than 0.5.

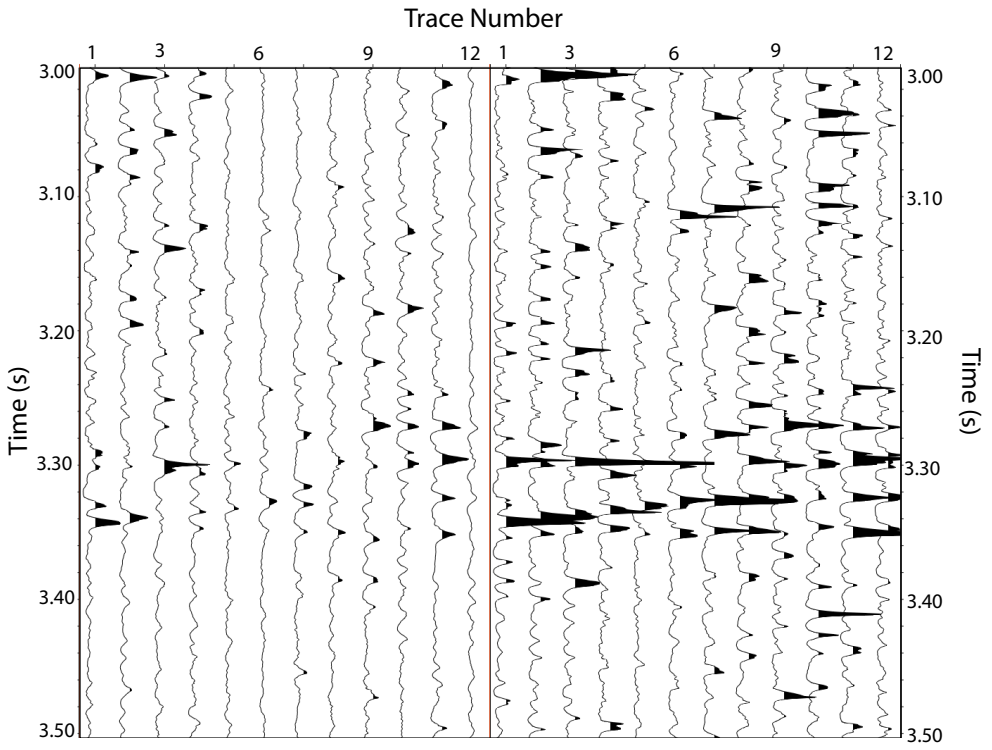


Figure C.10: Left panel: shows the result of calculating the ratios of the average unweighted linearity in window 4 between two moving windows with sizes of 7 ms. Right panel: shows the result of calculating the ratios of the average weighted linearity in window 4 between two moving windows with sizes of 7 ms. The plots have global scaling between 0 and 4. The black shades represent peaks with ratios above 2.5.

It seems that there is only one medium event in the data (Figure C.8), however, the new, weighted linearity result (middle panel, Figure C.9(b)) shows two distinct events that are very close to each other. The earlier event (the barely noticeable event) has identifiable S and S coda waves, and a poorly noticeable P wave in the results of the weighted linearity method (middle panel, Figure C.9(b)). In the same result, the later event (the medium event) has very clear S and S coda waves, with a P wave that is mixed up with the S coda from the earlier event. The final result after applying the signal detect techniques is shown in the right panel (Figure C.9(b)), and it seems that I can identify 6 signals (see Figure C.11). They also can be traceable in the ratios plot (right panel, Figure C.10). In comparison, the unweighted linearity method (middle panel, Figure C.9(a)) and its final result (right panel, Figure C.9(a)) clearly show the S coda, parts of the S wave,

and none of the P wave for the earlier event. Furthermore, they show only the S coda and S wave for the later event, not the P wave.

In window 4, I can detect 6 signals using the new technique (see Figure C.11 and 4 events using the traditional, unweighted linearity method.

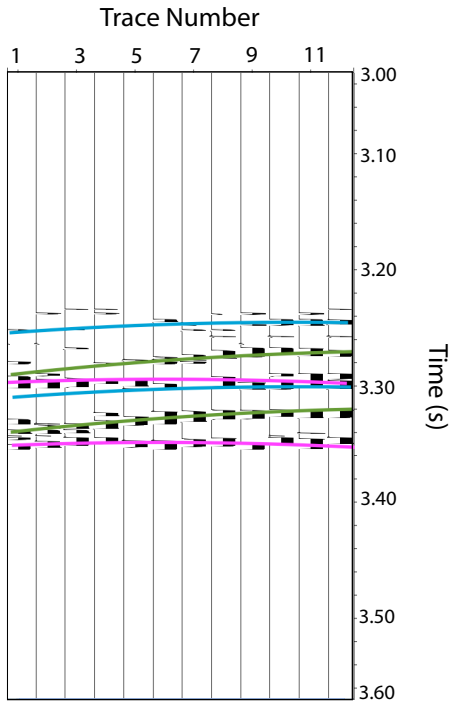


Figure C.11: Shows the signal detection result from the new weighted linearity method (middle panel, Figure C.9(b)). The blue, green, and magenta curves represent the P, S, and S coda waves, respectively.

C.2.5 Window 5 (3.5 s - 5 s)

Figure C.12 shows a zoomed image of window 5, which includes no events. The unweighted and weighted linearity result are shown in Figures C.13(a) and C.13(b) along with the final result of signal detection.

Window 5 (Figure C.12) is the last window in the 5 s fracture file, which covers no events. The unweighted linearity method (middle panel, Figure C.13(a)) and its final result of signal detect

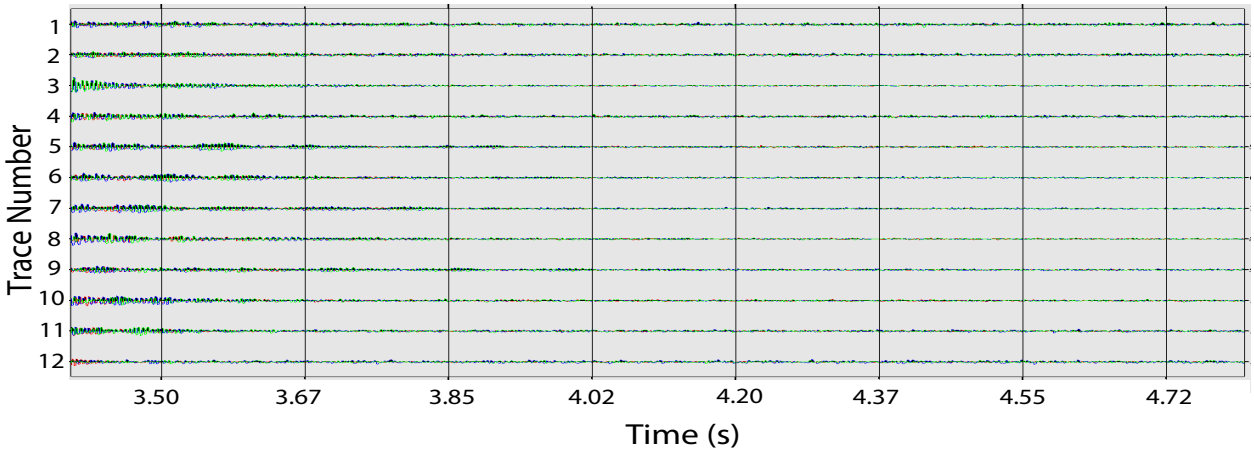
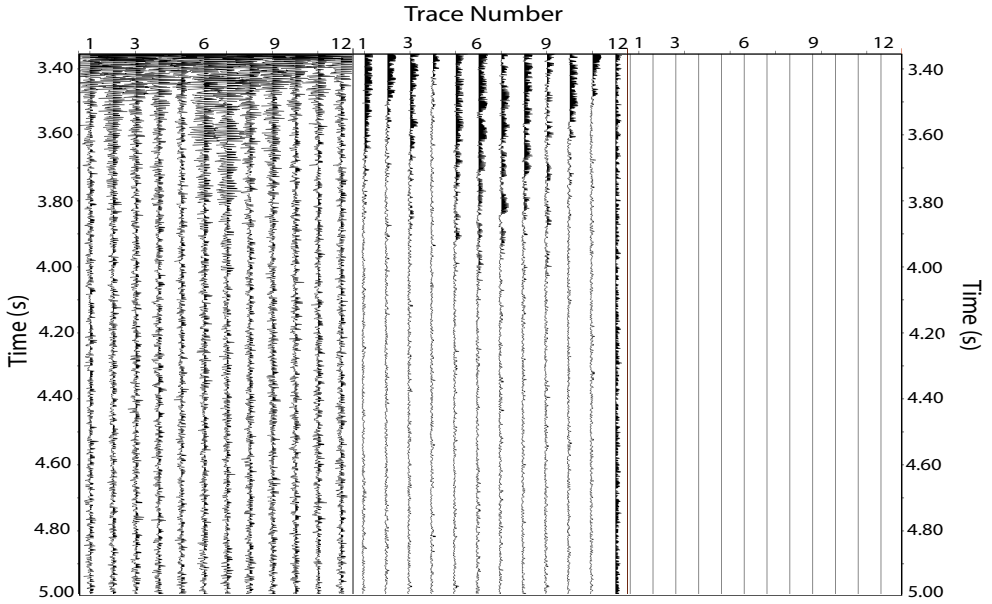
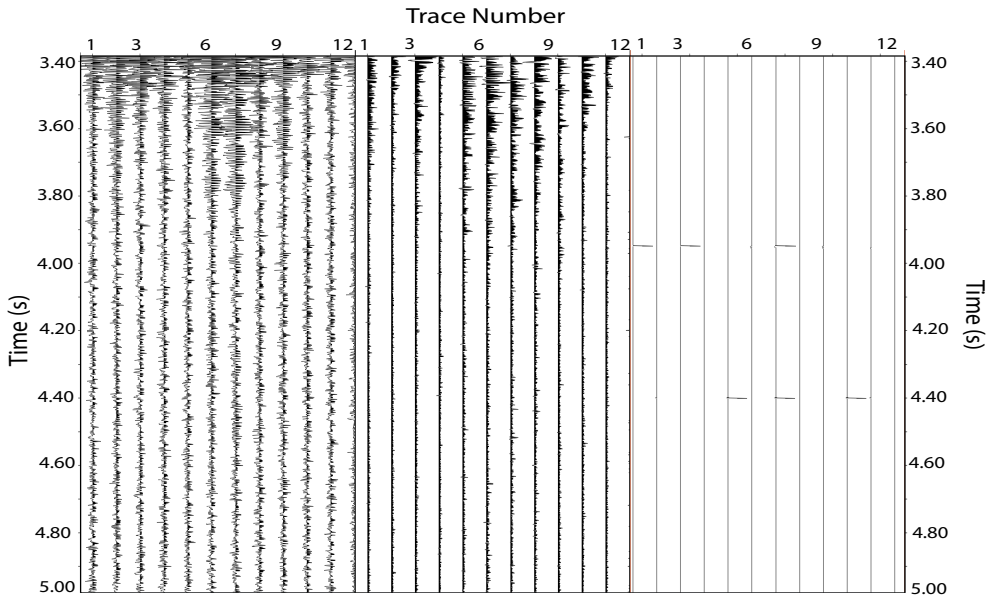


Figure C.12: Shows a zoomed plot of Figure 4.3 at window 5. The different waveforms' color represents the different component of a geophone, where red, blue and green are Z, X, and Y, respectively.

code (right panel, Figure C.13(a)) show no events. The weighted linearity method (middle panel, Figure C.13(b)) show no events as well, and the final result (right panel, Figure C.13(b)) has only a few peaks with no clear events. These peaks could be an indication of a very weak signals (since they have a straight-line shape), or they could be noise.



(a)



(b)

Figure C.13: Left panels: show the vertical component of unfiltered data from window 5 Figure 4.3. Middle panels: a) shows the result of the unweighted linearity calculation, and the plot has global scaling between 0 and 1. b) shows the result of the weighted linearity calculation. The plot has global scaling between 0 and 0.3. Right panels: show the result of detecting signal from the middle panels, where the peaks (signals) have a value of 1 and the rest are zeros. The plots have global scaling between 0 and 1. The black shades represent peak with values higher than 0.5.

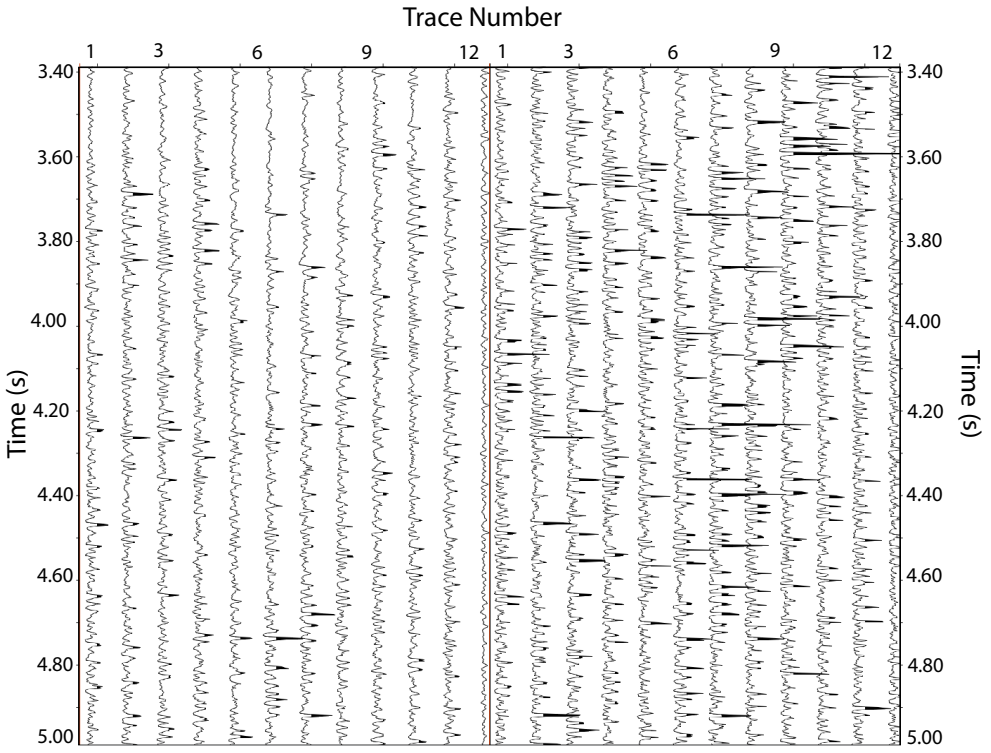


Figure C.14: Left panel: shows the result of calculating the ratios of the average unweighted linearity in window 5 between two moving windows with sizes of 7 ms. Right panel: shows the result of calculating the ratios of the average weighted linearity in window 5 between two moving windows with sizes of 7 ms. The plots have global scaling between 0 and 4. The black shades represent peaks with ratios above 2.5.

APPENDIX D

WEIGHTED AND UNWEIGHTED LINEARITY CODE

```
# This Script is encountering displacement information along with the three-dimensional time coherence analysis of the 3D particles motion (linearity).
#The method uses  four  different  approaches  of  multiplying  the  displacement  information  with the linearity results.
# Author: Abdullah Ahmed
# 12 November 2017
# Modify by: Milan Brankov & Abdullah Ahmed
# 12 February 2018
#============================================================================
#Reading Microseismic data using INTViewer functions
chooserA = SeismicDataChooser()
myDataA=chooserA.getSelectedData()

seismicWindowA=XSectionWindow("Data Set A")
seismicLayerA=SeismicLayer(seismicWindowA,myDataA)
seismicLayerA.setPlotType("wiggle")
seismicLayerA.setNormalizationType("RMS")
seismicLayerA.setNormalizationSync(True)
seismicLayerA.setEnableBroadcast(True)
seismicLayerA.setSeismicProcessorsSync(True)
#============================================================================
#Applying Bandpass filter to Microseismic data using INTViewer function
bandpass = StandaloneSeismicProcessor(myDataA, "Filter")
bandpass.setParameterValue('Freq01','10')
bandpass.setParameterValue('Freq11','20')
bandpass.setParameterValue('Freq21','250')
bandpass.setParameterValue('Freq31','300')
#============================================================================
import math
import cmath
#BELOW ARE FUNCTIONS THAT WE USE FOR FAST FOURIER TRANSFORM

def fourierm(x):
    N=len(x)
    y=[0]*N
    for i in range(0, N):
        y[i]=0
        for j in range(0,N):
            y[i]=y[i]+x[j]*cmath.exp(2*math.pi*1j*i*j/N)
        y[i]=y[i]*2 / N
    return y
```

```
40
41  def fastfourierm(x):
42      N=len(x)
43      i=2
44      while i<N**0.5+1:
45          if N%i==0:
46              N=0
47          else:
48              i=i+1
49      if N==0:
50          N=len(x)
51          y=[0]*N
52          c=int(N/i)
53          A = [[0] * c for row in range(i)]
54          B = [[0] * c for row in range(i)]
55          for j in range (0,i):
56              for k in range(0,c):
57                  A[j][k]=x[k*i+j]
58              B[j]=fastfourierm(A[j])
59          for m in range (0,N):
60              for n in range (0,i):
61                  y[m]=y[m]+B[n][m%c]*cmath.exp(1j*2*math.pi*n*m/N)
62              y[m]=y[m]/i
63      else:
64           return fourierm(x)
65      return y
66   #BELOW ARE FUNCTIONS THAT WE USE FOR FAST INVERSE FOURIER TRANSFORM
67
68  def fastinvfourierm(y):
69      N=len(y)
70      z=[0]*N
71      z2=fastinvfourierm2(y)
72      for i in range(0, N):
73          z[i]=z[i]+z2[i].real
74   #       z[i]=round(z[i],1)
75      return z
76
77  def fastinvfourierm2(y):
78      N=len(y)
79      i=2
80      while i<N**0.5+1:
81          if N%i==0:
82              N=0
83          else:
84              i=i+1
85      if N==0:
86          N=len(y)
87          z=[0]*N
88          c=int(N/i)
89          A = [[0] * c for row in range(i)]
90          B = [[0] * c for row in range(i)]
91          for j in range (0,i):
92              for k in range(0,c):
93                  A[j][k]=y[k*i+j]
94              B[j]=fastinvfourierm2(A[j])
95          for m in range (0,N):
96              for n in range (0,i):
97                  z[m]=z[m]+B[n][m%c]*cmath.exp(-1j*2*math.pi*n*m/N)
98      else:
99           return invfourierm2(y)
100     return z
101
102 def invfourierm2(y):
103     N=len(y)
104     z=[0]*N
105     for i in range(0, N):
106         for j in range(0,N):
107             v=y[j]*cmath.exp(-2*math.pi*1j*i*j/N)
108             z[i]=z[i]+0.5*v
109     return z
110  #BELOW ARE FUNCTIONS THAT WE USE TO DEFINE THE HILBERT TRANSFORM AND THE ENVELOPE
111
112 def hilbert(y):
113     N=len(y)
114     yh=[0]*N
115     for i in range (1,N):
116         if i<N/2:
117             yh[i]=y[i]*(-1j)
118         else:
119             yh[i]=y[i]*1j
120     return yh
```

```
121
122 def envelope(x):
123     y=fastfourierm(x)
124     z=fastinvfourierm(y)
125     yh=hilbert(y)
126     zh=fastinvfourierm(yh)
127     N=len(x)
128     env=[0]*N
129     for i in range (0,N):
130         env[i]=(z[i]**2+zh[i]**2)**0.5
131     return env
132  # Functions below to modify the envelope by adding the dervative
133 def env_modify(envx,envy,envz):
134     N=len(envx)
135     env=[0]*N
136     envd=[0]*N
137     for i in range (0,N):
138         env[i]=(envx[i]**2+envy[i]**2+envz[i]**2)**0.25
139     for i in range (1,N-1):
140         envd[i]=env[i+1]-env[i-1]
141     s1=0
142     s2=0
143     for i in range(0, N):
144         s1=s1+env[i]
145         s2=s2+abs(envd[i])
146     s=s1/s2
147     for i in range(0, N):
148         env[i]=env[i]+envd[i]*s
149     return env
150
151 def env_new(envx,envy,envz):
152     N=len(envx)
153     env=[0]*N
154     envd=[0]*N
155     for i in range (0,N):
156         env[i]=(envx[i]**2+envy[i]**2+envz[i]**2)**0.25
157     for i in range (1,N-1):
158         envd[i]=((envx[i+1]-envx[i-1])**2+(envy[i+1]-envy[i-1])**2+(envz[i+1]-envz[i-1])**2)**0.25
159     s1=0
160     s2=0
161     for i in range(0, N):
162         s1=s1+env[i]
163         s2=s2+envd[i]
164     s=s1/s2
165     envf=[0]*N
166     for i in range(0, N):
167         envf[i]=env[i]+envd[i]*s
168     return envf
169
```

```
#================================================================================
#ns is the number of samples within a window
#dT is the number of samples, were the window is moving
ns = 16
dT = 1

#Reading trace samples using INTViewer function
query = XSectionRangeQuery("TraceNumber", 1, 37)
seismicReader = myDataA.select(query)

# open files to write results

output = open('/tmp/temp_seismic_3.txt','w') # approach 3
output1 = open('/tmp/temp_seismic_origional_lin.txt','w') # origional linearity
output2 = open('/tmp/temp_seismic_2.txt','w')# approach 2
output3 = open('/tmp/temp_seismic_1.txt','w') # approach 1
output4 = open('/tmp/temp_seismic_4.txt','w') # approach 4
output5 = open('/tmp/temp_seismic_envelope.txt','w') # envelope



#================================================================================
# start to a loop to read the three component for each geophone
for trace in range (1,36,3):
    Mylist=[]
    Mylist1=[]
    Mylist2=[]
    Mylist3=[]
    Mylist4=[]
    Mylist5=[]
    Mylist6=[]
    Mylist7=[]
    Mylist8=[]
    Mylist9=[]
    Mylist10=[]
    Mylist11=[]



    XTrace = seismicReader.getTrace(trace).getSamples()
    YTrace = seismicReader.getTrace(trace+1).getSamples()
    ZTrace = seismicReader.getTrace(trace+2).getSamples()

    XTrace1 = seismicReader.getTrace(trace)
    YTrace1 = seismicReader.getTrace(trace+1)
    ZTrace1 = seismicReader.getTrace(trace+2)

    bandpass.processTrace(XTrace1)
    bandpass.processTrace(YTrace1)
    bandpass.processTrace(ZTrace1)
    XTrace2 = XTrace1.getSamples()
    YTrace2 = YTrace1.getSamples()
    ZTrace2 = ZTrace1.getSamples()

```

```
    envx = envelope(XTrace2)
    envy = envelope(YTrace2)
    envz = envelope(ZTrace2)
    env=env_modify(envx,envy,envz)



#iSamples is the number of samples in the trace and dT is the number of samples for the moving window
    for iSamples in range(0,2000-ns,dT):
    #print ("iSamples = "), iSamples

#read Trace X
        samplesX= XTrace[iSamples:iSamples+ns]
#read Trace Y
        samplesY = YTrace[iSamples:iSamples+ns]
#read Trace Z
        samplesZ = ZTrace[iSamples:iSamples+ns]

        center=int(ns/2)

    Mylist5.append(env)


#================================================================================
# calculating the mean
        meanX= sum(samplesX)/float(len(samplesX))

        meanY= sum(samplesY)/float(len(samplesY))

        meanZ= sum(samplesZ)/float(len(samplesZ))

        lenX=int(len(samplesX))
        print ("lenX = "), lenX
#================================================================================
# calculating the variance and Covariance

#nSamples is the number of samples within a window
        if iSamples = 0:

        d=0
        for nSamples in range (5,13):
        d=d+((samplesX[nSamples])**2+(samplesY[nSamples])**2+(samplesZ[nSamples])**2)**0.5
        d1=d/8

            Cxx=0
            Cxy=0
            Cxz=0
            Cyy=0
            Cyz=0
            Czz=0

            for nSamples in range (0,lenX):
        Cxx = Cxx + ((samplesX[nSamples]-meanX)*(samplesX[nSamples]-meanX))
            Cxx = Cxx/lenX
```

```
278
279             for nSamples in range (0,lenX):
280         Cxy = Cxy + ((samplesX[nSamples]-meanX)*(samplesY[nSamples]-meanY))
281             Cxy = Cxy/lenX
282
283             for nSamples in range (0,lenX):
284                 Cxz = Cxz + ((samplesX[nSamples]-meanX)*(samplesZ[nSamples]-meanZ))
285             Cxz = Cxz/lenX
286
287             for nSamples in range (0,lenX):
288                 Cyy = Cyy + ((samplesY[nSamples]-meanY)*(samplesY[nSamples]-meanY))
289             Cyy = Cyy/lenX
290
291             for nSamples in range (0,lenX):
292                 Cyz = Cyz + ((samplesY[nSamples]-meanY)*(samplesZ[nSamples]-meanZ))
293             Cyz = Cyz/lenX
294
295             for nSamples in range (0,lenX):
296                 Czz = Czz + ((samplesZ[nSamples]-meanZ)*(samplesZ[nSamples]-meanZ))
297             Czz = Czz/lenX
298             X0 = meanX
299             Y0 = meanY
300             Z0 = meanZ
301         else:
302             delx = meanX - X0
303             dely = meanY - Y0
304             delz = meanZ - Z0
305
306         d1=d1 - ((samplesX[4])**2+(samplesY[4])**2+(samplesZ[4])**2)**0.5/8 + ((samplesX[13])**2+(samplesY[13])**2+(samplesZ[13])**2)**0.5/8
307
308         Vxx = Cxx * ns + delx*delx
309         Cxx=(Vxx - (XTrace[iSamples-1]- meanX)**2 + ((XTrace[iSamples+ns]-meanX)**2))/ns
310         Vxy = Cxy * ns + delx*dely
311         Cxy=(Vxy - (XTrace[iSamples-1]- meanX)* (YTrace[iSamples-1]- meanY) + ((XTrace[iSamples+ns]-meanX)* (YTrace[iSamples+ns]-meanY)))/ns
312         Vxz = Cxz * ns + delx*delz
313         Cxz=(Vxz - (XTrace[iSamples-1]- meanX)* (ZTrace[iSamples-1]- meanZ) + ((XTrace[iSamples+ns]-meanX)* (ZTrace[iSamples+ns]-meanZ)))/ns
314         Vyy = Cyy * ns + dely*dely
315         Cyy=(Vyy - (YTrace[iSamples-1]- meanY)* (YTrace[iSamples-1]- meanY) + ((YTrace[iSamples+ns]-meanY)* (YTrace[iSamples+ns]-meanY)))/ns
316         Vyz = Cyz * ns + dely*delz
317         Cyz=(Vyz - (YTrace[iSamples-1]- meanY)* (ZTrace[iSamples-1]- meanZ) + ((YTrace[iSamples+ns]-meanY)* (ZTrace[iSamples+ns]-meanZ)))/ns
318         Vzz = Czz * ns + delz*delz
319         Czz=(Vzz - (ZTrace[iSamples-1]- meanZ)* (ZTrace[iSamples-1]- meanZ) + ((ZTrace[iSamples+ns]-meanZ)* (ZTrace[iSamples+ns]-meanZ)))/ns
320             X0 = meanX
321             Y0 = meanY
322             Z0 = meanZ
323
```

```
#=====================================================================
# calculating the average amplitudes

    dxx=0
    dyy=0
    dzz=0
    for nSamples in range (0,lenX):

        dxx = dxx + ((samplesX[nSamples])*(samplesX[nSamples]))/lenX

    for nSamples in range (0,lenX):

        dyy = dyy + ((samplesY[nSamples])*(samplesY[nSamples]))/lenX

    for nSamples in range (0,lenX):

        dzz = dzz + ((samplesZ[nSamples])*(samplesZ[nSamples]))/lenX
#=====================================================================
# calculating the eigenvalues
        a= -1
        b= (Cxx+Cyy+Czz)
        c= ((Cxz*Cxz)+(Cxy*Cxy)+(Cyz*Cyz)-(Cxx*Cyy)-(Cxx*Czz)-(Cyy*Czz))
        d= ((Cxx*Cyy*Czz)-(Cxx*Cyz*Cyz)-(Cxy*Cxy*Czz)-(Cxz*Cxy*Cyz)+(Cxz*Cxy*Cyz)+(Cxz*Cxz*Cyy))

        print ("a="),a
        print ("b="),b
        print ("c="),c
        print ("d="),d



        part1_lamda1= -b/(3*a)
        part2_lamda1= complex((((2**(1.0/3.0))*((-b*b)+(3*a*c)))/(3*a*(((-2*b*b*b)+(9*a*b*c)-(27*a*a*d)+(((4*((-b*b+3*a*c)**3.0))+(((-2*b*b*b)+(9*a*b*c)-(27*a*a*d))**2)+0j)**(0.5)))**(1.0/3)))))).real


        part3_lamda1= complex((((((-2*b*b*b)+(9*a*b*c)-(27*a*a*d)+(((4*((-b*b+3*a*c)**3.0))+(((-2*b*b*b)+(9*a*b*c)-(27*a*a*d))**2.0)+0j)**(0.5)))**(1.0/3.0)))/(3*a*(2**(1.0/3.0))))).real


        lamda1=part1_lamda1-part2_lamda1+part3_lamda1


        print ("lamda1="),lamda1


        part1_lamda2= -b/(3*a)
        part2_lamda2=complex((((1+(1j*3**(0.5)))*(-(b*b)+(3*a*c)))/(3*(2**(2.0/3.0))*a*(((-2*b*b*b)-(27*a*a*d)+(9*a*b*c)+(((4*((-b*b)+(3*a*c))*((-b*b)+(3*a*c))*((-b*b)+(3*a*c)))+(((-2*b*b*b)+(9*a*b*c)-(27*a*a*d))**2.0)+0j)**(0.5)))**(1.0/3.0))))).real


        part3_lamda2=complex((((1-1j*3**(0.5)))*((-2*b*b*b)-(27*a*a*d)+(9*a*b*c)+(((4*((-b*b)+(3*a*c))*((-b*b)+(3*a*c))*((-b*b)+(3*a*c)))+(((-2*b*b*b)+(9*a*b*c)-(27*a*a*d))**2.0)+0j)**(0.5)))**(1.0/3.0))/(6*a*(2**(1.0/3.0))))).real


        lamda2=part1_lamda2+part2_lamda2-part3_lamda2

        print ("lamda2="),lamda2


        part1_lamda3= -b/(3*a)

        part2_lamda3=complex((((1-1j*3**(0.5)))*(-(b*b)+(3*a*c)))/(3*(2**(2.0/3.0))*a*(((-2*b*b*b)-(27*a*a*d)+(9*a*b*c)+(((4*((-b*b)+(3*a*c))*((-b*b)+(3*a*c))*((-b*b)+(3*a*c)))+(((-2*b*b*b)+(9*a*b*c)-(27*a*a*d))**2.0)+0j)**(0.5)))**(1.0/3.0))))).real


        part3_lamda3=complex((((1+1j*3**(0.5)))*((-2*b*b*b)-(27*a*a*d)+(9*a*b*c)+(((4*((-b*b)+(3*a*c))*((-b*b)+(3*a*c))*((-b*b)+(3*a*c)))+(((-2*b*b*b)+(9*a*b*c)-(27*a*a*d))**2.0)+0j)**(0.5)))**(1.0/3.0))/(6*a*(2**(1.0/3.0))))).real



        lamda3=part1_lamda3+part2_lamda3-part3_lamda3

        print ("lamda3="),lamda3
```

```
#================================================================================
# Global polarization
        Cp0= (((lamda1-lamda2)**2)+((lamda1-lamda3)**2)+((lamda2-lamda3)**2))/(2*((lamda1+lamda2+lamda3)**2))
# multiplying the four appraoches with the global polarization
        Cp=Cp0*(d1**0.5)
    Cp1=Cp0*d1
    Cp2=Cp0*((dxx+dyy+dzz)**0.5)
    Cp3=Cp0*env


        print ("Cp="),Cp
        Mylist.append(Cp)
    Mylist1.append(Cp0)
    Mylist2.append(Cp1)
    Mylist3.append(Cp2)
    Mylist4.append(Cp3)



#================================================================================
#saving the Global polarization


    for x in Mylist:
    output.write("{}\n".format(x))
    for k in Mylist1:
    output1.write("{}\n".format(k))
    for s in Mylist2:
    output2.write("{}\n".format(s))
    for f in Mylist3:
    output3.write("{}\n".format(f))
    for o in Mylist4:
    output4.write("{}\n".format(o))
    for q in Mylist5:
    output5.write("{}\n".format(q))



output.close()
output1.close()
output2.close()
output3.close()
output4.close()
output5.close()

# converting text files into SU files using seismic unix commands
import commands

commands.getstatusoutput('a2b n1=1 </tmp/temp_seismic_3.txt >/tmp/temp_seismic_3.bin')
commands.getstatusoutput('a2b n1=1 </tmp/temp_seismic_origional_lin.txt >/tmp/temp_seismic_origional_lin.bin')
commands.getstatusoutput('a2b n1=1 </tmp/temp_seismic_2.txt >/tmp/temp_seismic_2.bin')
commands.getstatusoutput('a2b n1=1 </tmp/temp_seismic_1.txt >/tmp/temp_seismic_1.bin')
commands.getstatusoutput('a2b n1=1 </tmp/temp_seismic_4.txt >/tmp/temp_seismic_4.bin')
commands.getstatusoutput('a2b n1=1 </tmp/temp_seismic_envelope.txt >/tmp/temp_seismic_envelope.bin')


commands.getstatusoutput('suaddhead ns=1984 </tmp/temp_seismic_3.bin  | sushw key=dt a=250 > /tmp/temp_seismic_3.su')
commands.getstatusoutput('suaddhead ns=1984 </tmp/temp_seismic_origional_lin.bin  | sushw key=dt a=250 > /tmp/temp_seismic_origional_lin.su')
commands.getstatusoutput('suaddhead ns=1984 </tmp/temp_seismic_2.bin  | sushw key=dt a=250 > /tmp/temp_seismic_2.su')
commands.getstatusoutput('suaddhead ns=1984 </tmp/temp_seismic_1.bin  | sushw key=dt a=250 > /tmp/temp_seismic_1.su')
commands.getstatusoutput('suaddhead ns=1984 </tmp/temp_seismic_4.bin  | sushw key=dt a=250 > /tmp/temp_seismic_4.su')
commands.getstatusoutput('suaddhead ns=1984 </tmp/temp_seismic_envelope.bin  | sushw key=dt a=250 > /tmp/temp_seismic_envelope.su')
```

APPENDIX E

SIGNAL DETECT CODE

```
# This Script is detecting signals from the weighted and unweighted linearity calculations
# Author: Abdullah Ahmed
# 15 March 2018
# Modify by: Milan Brankov & Abdullah Ahmed
# 5 April 2018
#================================================================================
# Below function calculates the ratio between two moving windows
def avg_diff_norm(x,w1,w2):
    N=len(x)
    y=[0]*N
    s1=0
    s2=0
    for j in range (0,w1):
        s1=s1+abs(x[j])
    for j in range (w1+1,w1+w2+1):
        s2=s2+abs(x[j])
    y[w1]=(s2/w2)/(s1/w1)#*((s2/w2)+(s1/w1))**0.5
    for i in range (w1+1,N-w2):
        s1=s1-abs(x[i-w1-1])+abs(x[i-1])
        s2=s2-abs(x[i])+abs(x[i+w2])
        y[i]=(s2/w2)/(s1/w1)
    s=0
    for i in range (0,N):
       s=s+y[i]
    s=s/N
    y[0:w1]=[s]*w1
    y[N-w2:N]=[s]*w2
    return y
# Below function applies the cutoff value and compare high peaks distribution in adjacent
#traces to remove peaks introduced by noise
def purify(y,c,t): #y= input, C = cutoff value, t = width of a window where you want to look for adjacent peaks
     N=len(y)
     x=[0]*N
     for i in range(0,N):
     x[i]=y[i]
     for i in range (0,N):
     if x[i]>c:
         m=0
         if i+t+15<N:
         for j in range (i+t-15,i+t+15):
             if x[j]>m:
             m=x[j]
         else:
                 for j in range (i-2*t-15,i-2*t+15):
                     if x[j]>m:
                         m=x[j]
         if i-t-15>0:
                 for j in range (i-t-15,i-t+15):
                     if x[j]>m:
                         m=x[j]
         else:
         for j in range (i+2*t-15,i+2*t+15):
                     if x[j]>m:
                         m=x[j]
```

```
        if m<c:
                x[i]=c*0.9
    for i in range (0,N):
       if x[i]<c:
           x[i]=0
       else:
           x[i]=1
    return x

# Below function calculates the geometric average
def geom_avg(y,c,t,w1,w2):#y= input, C = cutoff value, t = width of a window where you want to look for adjacent peaks
                          #w1 = length of window w2 = length  of window 2
    N=len(y)
    x=[0]*N
    for i in range(0,N):
    x[i]=y[i]
    for i in range (0,N):
    if x[i]>c:
        m=0
        if i+t+w1<N:
        for j in range (i+t-w1,i+t+w1):
            if x[j]>m:
            m=x[j]
        else:
        for j in range (i-2*t-w1,i-2*t+w1):
            if x[j]>m:
            m=x[j]
        if i-t-w1>0:
        for j in range (i-t-w1,i-t+w1):
            if x[j]>m:
            m=x[j]
        else:
        for j in range (i+2*t-w1,i+2*t+w1):
            if x[j]>m:
            m=x[j]
        if m<c:
        x[i]=c*0.99
    mult=[1]*N
    for i in range (0,N):
    if x[i]>c:
        m=0
        if i+t+w2<N:
        for j in range (i+t-w2,i+t+w2):
            if x[j]>m:
            m=x[j]
        mult[i]=mult[i]*(m/x[i])**0.25
        m=0
        if i-t-w2>0:
        for j in range (i-t-w2,i-t+w2):
            if x[j]>m:
            m=x[j]
        mult[i]=mult[i]*(m/x[i])**0.25
    else:
        m=0
        if i+t+w2<N:
        for j in range (i+t-w2,i+t+w2):
            if x[j]>m:
            m=x[j]
        if m>c:
            mult[i]=mult[i]*(m/x[i])**0.25
        m=0
        if i-t-w2>0:
        for j in range (i-t-w2,i-t+w2):
            if x[j]>m:
            m=x[j]
        if m>c:
            mult[i]=mult[i]*(m/x[i])**0.25
    for i in range (0,N):
    x[i]=x[i]*mult[i]
    return x
# This function stack all 12 traces
def stackk(x,w,t):#x= input, w = size of the window you want to stack, t = time where the window is moving
    N = len(x)
    n=N/t
    sc=[0]*t
    for i in range (0+w,t-w):
    for j in range (0,n):
        for k in range (-w,w):
        sc[i]=sc[i]+x[i+t*j+k]
    sc[0:w]=[sc[w]]*w
    sc[t-w:t]=[sc[t-w-1]]*w
    return sc
# This function applies the cutoff value
def cutoff(x,y):
    N=len(x)
    t=len(y)
    n=N/t
    m=0
    for i in range(0,t):
    if m<y[i]:
        m=y[i]
    z=[0]*N
    for i in range (0,N):
    z[i]=x[i]
    for i in range (0,n):
    for j in range (0,t):
        if y[j]<m/2:
        z[i*t+j]=0
    return z
```

```
# This function calculates the average within a window
def average(x):
    N = len(x)
    y1 = 0
    y2 = 0
    for i in range(0,N):
        y1=y1+x[i]
        y2=y2+(x[i])**2
    y1tot=y1/N
    y2tot=y2/N
    return [y1tot,y2tot]

#def cutsection(xo,z,t):
    #N=len(xo)
    #x=[0]*N
    #for i in range (0,N):
        #x[i]=xo[i]
    #n=N/t
    #y=stackk(z,15,t)
    #for i in range (0,t):
    #if y[i]>0.5:
        #y[i]=1
    #else:
        #y[i]=0
    #for i in range (0,t):
    #if y[i]<0.5:
        #for j in range (0,n):
        #x[i+j*t]=0
    #return x

#def deletejunk(x,z,ct):
    #N=len(x)
    #y=[0]*N
    #for i in range (0,N):
    #y[i]=z[i]
    #for i in range (0,N):
    #if y[i]<ct:
        #y[i]=0
    #else:
        #y[i]=1
    #avg=0
    #for i in range (0,N):
    #avg=avg+x[i]
    #avg=avg/N
    #for i in range (0,N):
    #if y[i]>0.5:
        #a=0
        #for j in range (i-3,i+4):
        #a=a+x[j]
        #if a/2>avg:
        #y[i]=0
    #return y
# This function applies a vertical shift, which calculated by the cross-correlation
def shifteachtrace(x,t,sh):#x= input, t = time where the window is moving, sh = time shift
    N=len(x)
    z=[0]*N
    for i in range (0,N):
    z[i]=x[i]
    n=int(N/t)
    for i in range (0,n):
    a=0
    for j in range (0,t):
        a=a+x[j+i*t]
    a=a/t
    for j in range (0,sh):
        z[j+i*t]=a
    for j in range (sh,t):
        z[j+i*t]=x[j+i*t-sh]
    return z

#BELOW ARE FUNCTIONS THAT WE USE FOR FAST FOURIER TRANSFORM
import cmath
import math
def fourierm(x):
    N=len(x)
    y=[0]*N
    for i in range(0, N):
        y[i]=0
        for j in range(0,N):
            y[i]=y[i]+x[j]*cmath.exp(2*math.pi*1j*i*j/N)
        y[i]=y[i]*2 / N
    return y

def fastfourierm(x):
    N=len(x)
    i=2
    while i<N**0.5+1:
        if N%i==0:
            N=0
        else:
            i=i+1
    if N==0:
        N=len(x)
        y=[0]*N
        c=int(N/i)
        A = [[0] * c for row in range(i)]
        B = [[0] * c for row in range(i)]
        for j in range (0,i):
            for k in range(0,c):
                A[j][k]=x[k*i+j]
            B[j]=fastfourierm(A[j])
        for m in range (0,N):
            for n in range (0,i):
                y[m]=y[m]+B[n][m%c]*cmath.exp(1j*2*math.pi*n*m/N)
```

```
                    y[m]=y[m]+B[n][m%c]*cmath.exp(1j*2*math.pi*n*m/N)
            y[m]=y[m]/i
    else:
        return fourierm(x)
    return y

#BELOW ARE FUNCTIONS THAT WE USE FOR FAST INVERSE FOURIER TRANSFORM
def fastinvfourierm(y):
    N=len(y)
    z=[0]*N
    z2=fastinvfourierm2(y)
    for i in range(0, N):
        z[i]=z[i]+z2[i].real
#         z[i]=round(z[i],1)
    return z

def fastinvfourierm2(y):
    N=len(y)
    i=2
    while i<N**0.5+1:
        if N%i==0:
            N=0
        else:
            i=i+1
    if N==0:
        N=len(y)
        z=[0]*N
        c=int(N/i)
        A = [[0] * c for row in range(i)]
        B = [[0] * c for row in range(i)]
        for j in range (0,i):
            for k in range(0,c):
                A[j][k]=y[k*i+j]
            B[j]=fastinvfourierm2(A[j])
        for m in range (0,N):
            for n in range (0,i):
                z[m]=z[m]+B[n][m%c]*cmath.exp(-1j*2*math.pi*n*m/N)
    else:
        return invfourierm2(y)
    return z

def invfourierm2(y):
    N=len(y)
    z=[0]*N
    for i in range(0, N):
        for j in range(0,N):
            v=y[j]*cmath.exp(-2*math.pi*1j*i*j/N)
            z[i]=z[i]+0.5*v
    return z

# Below function finds the maximum peak in the cross-correlation result
def findmax(x):
    N=len(x)
    m=x[0]
    sh=0
    for i in range (0,N):
        if x[i]>m:
            m=x[i]
            sh=i
    if sh>N/2:
        sh=N-sh
    return sh
#================================================================================
#================================================================================
#Reading Microseismic data using INTViewer functions
chooserA = SeismicDataChooser()
myDataA=chooserA.getSelectedData()

seismicWindowA=XSectionWindow("Data Set A")
seismicLayerA=SeismicLayer(seismicWindowA,myDataA)
seismicLayerA.setPlotType("wiggle")
seismicLayerA.setNormalizationType("RMS")
seismicLayerA.setNormalizationSync(True)
seismicLayerA.setEnableBroadcast(True)


#Reading trace samples using INTViewer function
query = XSectionRangeQuery("TraceNumber", 1, 13)
seismicReader = myDataA.select(query)

# open files to write results

output = open('/tmp/temp_seismic_signal_detect_ave_ratio.txt','w') # output the ratio plots
output1 = open('/tmp/temp_seismic_signal_detect_purify.txt','w') # output the plots after applying the cutoff value
output2 = open('/tmp/temp_seismic_signal_detect_stack.txt','w') # stack the previous output 1
output3 = open('/tmp/temp_seismic_signal_detect_stack_cutoff.txt','w') # apply a cutoff from the stack into output 1


Mylist=[]
Mylist1=[]
Mystack=[]
Mylist2=[]
Mylist3=[]
Mylist4=[]
Mylist5=[]
for iSamples in range(0,12):

    Trace = seismicReader.getTrace(iSamples).getSamples()

    ave=avg_diff_norm(Trace,28,28)
    print len(ave)
    for i in range (0, len(ave)):
    Mylist.append(ave[i])
    for i in range (0, len(Trace)):
    Mylist4.append(Trace[i])
```

```
362
363  # applying Cross-correlation to figure out the vertical shift needed
364      XTraceC = fastfourierm(Trace)
365      XTraceZ = fastfourierm(ave)
366      XTraceD = []
367      N=len(XTraceZ)
368      f1= [0]*N
369      for o in range (0,N):
370      f1=XTraceZ[o]-(2*1j*XTraceZ[o].imag)
371      XTraceD.append(f1)
372
373
374      print len(XTraceC)
375      print len(XTraceD)
376
377      Xfinal=[0]*len(XTraceD)
378      for z in range (0, len(XTraceD)):
379      Xfinal[z]=XTraceC[z] * XTraceD[z]
380      CrossCorrelation = fastinvfourierm(Xfinal)
381      for i in range (0, len(CrossCorrelation)):
382      Mylist5.append(CrossCorrelation[i])
383  # applying stack on the cross-correlation result and then find the maximum value of the stack
384  cross_stack=stackk(Mylist5,1,1984)
385
386  MAX=findmax(cross_stack)
387
388  print MAX
389  # apply the shift, stack and cutoff
390  Mylist=shifteachtrace(Mylist,1984,MAX)
391
392  Mylistcutoff=stackk(Mylist,1,1984)
393
394  Mylist1=purify(Mylist,5,1984)
395  Mystack=stackk(Mylist1,1,1984)
396  Mylist2=cutoff(Mylist1,Mystack)
397  Mystack=stackk(Mylist2,100,1984)
398  Mylist2=cutoff(Mylist2,Mystack)
399
400
401  Y_average=average(Mylist)
402
403
404  print Y_average
```

```
405
406  #saving the results
407
408  for k in Mylist:
409      output.write("{}\n".format(k))
410  for n in Mylist1:
411      output1.write("{}\n".format(n))
412  for l in Mystack:
413      output2.write("{}\n".format(l))
414  for f in Mylist2:
415      output3.write("{}\n".format(f))
416  output.close()
417  output1.close()
418  output2.close()
419  output3.close()
420  output4.close()
421
422  # converting text files into SU files using seismic unix commands
423  import commands
424
425  commands.getstatusoutput('a2b n1=1 </tmp/temp_seismic_signal_detect_ave_ratio.txt >/tmp/temp_seismic_signal_detect_ave_ratio.bin')
426  commands.getstatusoutput('a2b n1=1 </tmp/temp_seismic_signal_detect_purify.txt >/tmp/temp_seismic_signal_detect_purify.bin')
427  commands.getstatusoutput('a2b n1=1 </tmp/temp_seismic_signal_detect_stack.txt >/tmp/temp_seismic_signal_detect_stack.bin')
428  commands.getstatusoutput('a2b n1=1 </tmp/temp_seismic_signal_detect_stack_cutoff.txt >/tmp/temp_seismic_signal_detect_stack_cutoff.bin')
429
430  commands.getstatusoutput('suaddhead ns=1984 </tmp/temp_seismic_signal_detect_ave_ratio.bin  | sushw key=dt a=250 > /tmp/temp_seismic_signal_detect_ave_ratio.su')
431  commands.getstatusoutput('suaddhead ns=1984 </tmp/temp_seismic_signal_detect_purify.bin  | sushw key=dt a=250 > /tmp/temp_seismic_signal_detect_purify.su')
432  commands.getstatusoutput('suaddhead ns=1984 </tmp/temp_seismic_signal_detect_stack.bin  | sushw key=dt a=250 > /tmp/temp_seismic_signal_detect_stack.su')
433  commands.getstatusoutput('suaddhead ns=1984 </tmp/temp_seismic_signal_detect_stack_cutoff.bin  | sushw key=dt a=250 > /tmp/temp_seismic_signal_detect_stack_cutoff.su')
434
```